9-1-2011

Towards flexible hardware/software encoding using H.264

Mark Hogan

Towards Flexible Hardware/Software Encoding using H.264

by

Mark D. Hogan

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science
in Computer Engineering

Supervised by

Dr. Marcin Łukowiak
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
September 2011

Approved by:

Dr. Marcin Łukowiak,
Thesis Advisor, Department of Computer Engineering

Dr. Andres Kwasinski,
Committee Member, Department of Computer Engineering

Dr. Michael Kurdziel,
Committee Member, Harris Corporation

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

Towards Flexible Hardware/Software Encoding using H.264

I, Mark D. Hogan, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Mark D. Hogan

Date

Dedication

To my God and my family.

Acknowledgments

I am grateful for Dr. Marcin Łukowiak's commitment to helping me complete this thesis and for his guidance and patience which were needed along the way. I'd also like to thank Dr. Michael Kurdziel and the Harris Corporation along with NYSTAR for their guidance and financial support of this work. Next, I'd like to thank Dr. Andres Kwasinski for his help with this thesis and the rest of the Computer Engineering department for teaching and preparing me for my future career. Furthermore, I'd like to thank Tim Sperr for mentoring me and allowing me to build off of his work. Last, but certainly not least, I'd like to give a special thanks to Richard Tolleson for supplying me with the lab equipment necessary for this thesis.

Abstract

Towards Flexible Hardware/Software Encoding using H.264

Mark D. Hogan

Supervising Professor: Dr. Marcin Łukowiak

As the electronics world continues to expand, bringing smaller and more portable devices to consumers, demands for media access continue to rise. Consumers are seeking the ability to view the wealth of information available on the Internet from devices such as smart phones, tablets, and music players. In addition to Internet browsing, smart phones and tablets in particular look to reinvent phone communication by adding video chat through services such as Skype and FaceTime. Bringing video to mobile platforms requires trade-offs between size, channel capacity, hardware cost, quality, loading times and power consumption. H.264, the current standard for video encoding specifies multiple profiles to support different modes of operation and environments. Creating an H.264 video encoder for a mobile platform requires a proper balance between the aforementioned trade-offs while maintaining flexibility in a real time environment such as video chatting.

The goal of this thesis was to investigate the trade-offs of implementing the H.264 Baseline encoding process specifically at low bit rates in hardware and software using Field Programmable Gate Array (FPGA) reconfigurable resources with an embedded processor core on the same chip. To further preserve encoding flexibility, existing encoding parameters were left intact. The Joint Model (JM) Reference encoder modified to include only the Baseline Profile was used as an initial reference point to evaluate the efficacy of the finished encoder. To improve upon the initial software implementation, major software bottlenecks were identified and hardware accelerators were designed aimed at producing a speedup capable of encoding 176x144 or Quarter Common Intermediate Format (QCIF) videos in real-time at 24

Frames Per Second (FPS) or greater. Finally, the hardware/software implementation was analyzed in comparison with the original JM Reference software encoder. This analysis included FPS, bit rate, encoding time, luminance Peak Signal-to-Noise Ratio (Y-PSNR) and associated hardware costs.

Contents

List of Tables

List of Figures

Glossary

chrominance – color difference, also known as chroma or C

I-Slice – composed of macroblocks that are predicted from data contained within the current frame. Also known as Intra-Slices

inter prediction – see motion estimation

intra prediction – prediction using blocks within the current slice

luminance – brightness, also known as luma or Y

macroblock – a fundamental building block used in video encoding. In H.264, this refers to a block corresponding to 16x16 pixel region of the source frame

median predictor – motion vector predicted from the median of the block to the left, top, and top-right

motion estimation – prediction using blocks from previous frames. Also known as inter prediction

P-Slice – contain macroblocks that are predicted using previously encoded reference frames. Also known as Predicted-Slices

residual – difference between the actual value and predicted value

slice – composed of interrelated macroblocks within a single frame

subsampling – utilizing fewer values than sampled

Acronyms

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

ASO Arbitrary Slice Order

AVC Advanced Video Coding

BRAM Block Random-Access Memory

CABAC Context-Adaptive Binary Arithmetic Coding

CAVLC Context-Adpative Variable-Length Coding

CLB Configurable Logic Block

CODEC Encoder/Decoder

DCT Discrete Cosine Transform

EDK Embedded Development Kit

EPZS Enhanced Predictive Zonal Search

FPGA Field Programmable Gate Array

FPS Frames Per Second

FPU Floating Point Unit

GPP General Purpose Processor

GPR General Purpose Register

HDL Hardware Description Language

HMDS Hardware-Oriented Modified Diamond Search

HVS Human Visual System

I/O Input/Output

ISO/IEC International Organization for Standardization / International Electrotechnical Commission

ITU-T International Telecommunication Union

JM Joint Model

JVT Joint Video Team

LED Light-Emitting Diode

LUT Look-Up Table

MPEG Moving Picture Experts Group

MSE Mean Squared Error

NAL Network Abstraction Layer

PLB Processor Local Bus

POSIX Portable Operating System Interface for Unix

PPC PowerPC

PPU Pixel Processing Unit

PSNR Peak Signal-to-Noise Ratio

QP Quantization Parameter

RBSP Raw Byte Sequence Payload

RDO Rate-Distortion Optimization

RGB Red, Green, Blue

RISC Reduced Instruction Set Computer

RoI Region of Interest

SAE Sum of Absolute Errors

SDK Software Development Kit

SRAM Static Random-Access Memory

UART Universal Asynchronous Receiver/Transmitter

UMHex Unsymmetrical-cross and Multi-Hexagon-grid

VCEG Video Coding Experts Group

VHDL Very-high-speed integrated circuit Hardware Description Language

VLC Variable Length Coding

XPS Xilinx Platform Studio

YCbCr Luminance, Blue Chrominance, Red Chrominance

Chapter 1

Introduction

H.264 offers the ability to stream video across networks with limited bandwidth. These networks highlight the need for a proper balance of quality and compression ratio. This trade-off of quality and compression ratio is important as the emphasis of quality versus compression changes based on the environment. For example, when using a device with limited storage space and a small screen such as a mobile phone, the quality of the video may not be as important as the ability to store many different videos at once and thus compression may be favored. In contrast, when encoding videos to be displayed on a media center PC with a large screen, the priority of quality over compression may be desirable. H.264 allows the specification of parameters to produce a video encoded according to the appropriate quality to compression proportion. To optimize H.264 encoding for environments with limited bandwidth, it is imperative to understand how the computational complexity of the encoder varies from stage to stage. Some of the stages are sequential in nature and thus lend themselves to software implementation while others are parallel in nature and are better suited for hardware. To achieve optimal encoding using the H.264 standard, a balance between hardware and software must be determined.

1.1 Project Description

This thesis looked to determine the feasibility of implementing the H.264 Baseline Profile using the Xilinx Ml-507 development board which includes a Virtex-5 Field Programmable Gate Array (FPGA) with an embedded PowerPC (PPC) 440 processor core. Software bottlenecks within the Joint Model (JM) Reference Baseline encoder were identified through research and profiling to determine their utility in a hardware/software system. The identified software bottlenecks were prioritized based on their percentage of execution time and suitability for hardware implementation. Software bottlenecks for which significant hardware speedup was expected were moved into hardware. By moving software bottlenecks into hardware, the time required to encode each frame was expected to decrease significantly from the time required using the software only encoder. One of the downsides of a purely hardware encoder is the loss of flexibility. Hardware encoders are usually designed with a specific task in mind and thus constructed in a way that the encoder performs one specific task efficiently. One of the goals of this thesis was to preserve the flexibility of the JM Reference Baseline software encoder despite moving portions of the encoding process into hardware. This allowed the hardware/software implementation to be flexible and tailored to many different scenarios. By maintaining the top level structure of a software encoder, the flexibility of the encoder was maintained in addition to obtaining the speed benefits of hardware accelerators in an effort to move toward a real time encoder.

The success of creating a flexible hardware/software H.264 encoder was evaluated through tests which compared the performance of the implemented encoder in contrast with the original JM Reference encoder. These tests show side-by-side comparisons of the software encoder with the hardware/software encoder. Specifically, the tests include the average number of Frames Per Second (FPS), bit rate, encoding time, luminance Peak Signal-to-Noise Ratio (Y-PSNR) and associated hardware costs.

1.2 Document Organization

The text is organized in a way that parallels the path of development taken to complete this thesis.

Chapter 2 provides an overview of the H.264 standard, Region of Interest (RoI) Encoding, and FPGAs. The goal of the chapter is to provide sufficient background to the reader such that the terminology used in the later chapters is coherent.

Chapter 3 looks at the contributions from previous efforts which promoted the research and developments of this thesis. The goal of the chapter is to provide the reader with knowledge of what has been done in the past and to further demonstrate the motivation for this thesis.

Chapter 4 details the design approach and resulting implementation. The goal of the chapter is to demonstrate how the flexible hardware/software H.264 encoder was constructed.

Chapter 5 describes the steps taken to verify functionality of the designed encoder and details the testing environment. The goal of this chapter is to explain the verification procedure and the testing setup.

Chapter 6 provides the results observed. The goal of the chapter is to show how the designed encoder compares with the original JM Reference encoder.

Chapter 7 presents conclusions and suggests opportunities for future work.

Chapter 2

Background

2.1 H.264 Video Coding Standard

This section provides an overview of the H.264 standard with an emphasis on components which directly apply to the text. H.264 or MPEG-4 Part 10 was designed by the Joint Video Team (JVT) formed by the combination of the Moving Picture Experts Group (MPEG) of the International Organization for Standardization / International Electrotechnical Commission (ISO/IEC) and the Video Coding Experts Group (VCEG) of the International Telecommunication Union (ITU-T). The standard was published as Advanced Video Coding (AVC) in 2003 under recommendation H.264 by the ITU-T and MPEG-4 Part 10 by the ISO/IEC [15]. H.264 improves upon previous MPEG and ITU-T standards with the addition of more efficient compression techniques.

2.2 H.264 Profiles

The H.264 standard defines multiple profiles for different use cases including the Baseline, Extended, Main, and High Profiles. Figure 2.1 shows a graphical representation of the features included in these profiles.

The Baseline Profile encodes videos using the core features of the H.264 standard. In addition, the Baseline Profile provides the ability to encode parts of the frame redundantly, allowing for additional error resilience. The Extended Profile builds on the features included in the Baseline Profile by adding data partitioning and additional compression techniques. The Main Profile includes the core functionality of the Baseline Profile, but does not

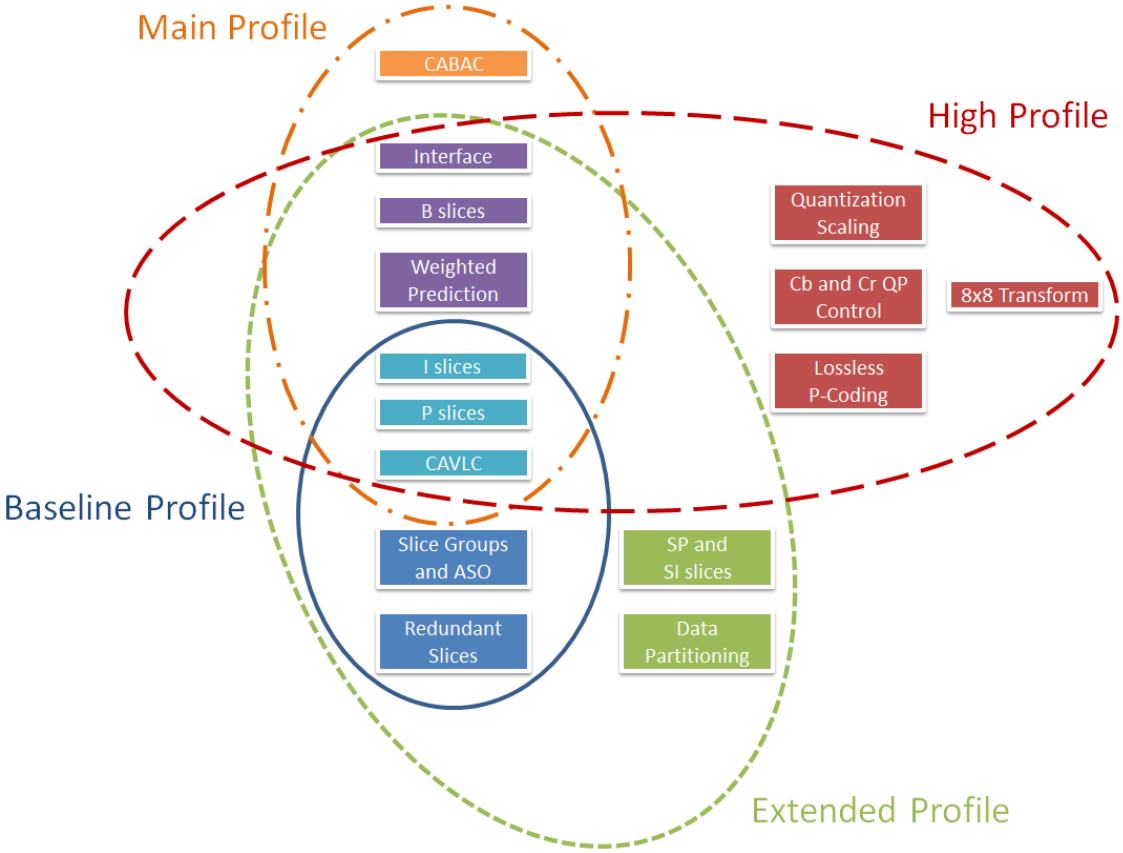


Figure 2.1: Four Primary H.264 Profiles [17]

include redundant partial frame encoding. The Main Profile also introduces interlacing support and increased levels of compression at the cost of higher memory and computational requirements. Lastly, the High Profile is built on top of the Main Profile and is designed for use with high definition and mainstream broadcast video.

2.3 H.264 Encoding Process

The H.264 standard matches the precedent set by previous video coding standards by defining the structure of the encoded bitstream rather than the format of the Encoder/Decoder (CODEC). Figure 2.2 shows a block diagram of the encoding process generally used for H.264. This section will break down the encoding process according to the blocks shown in Figure 2.2.

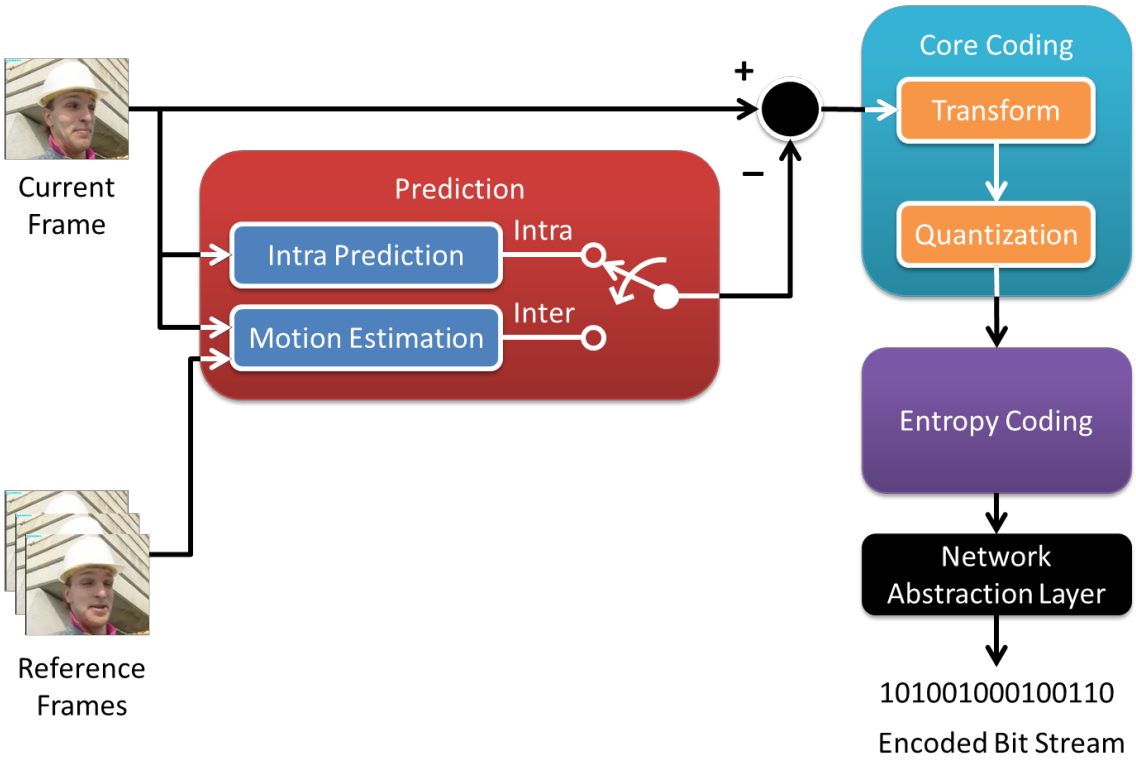


Figure 2.2: H.264 Encoder Block Diagram [4]

2.3.1 Fundamental Units

Video frames are broken down into fundamental units to facilitate computation and compression. Macroblocks and slices are formed from sets of pixels. Pixels are represented by values according to a predefined color space.

Macroblock

H.264 breaks each video frame into fundamental blocks called macroblocks. Macroblocks are formed from 16x16 pixel blocks of the original source frame. Figure 2.3 shows a frame separated into macroblocks.

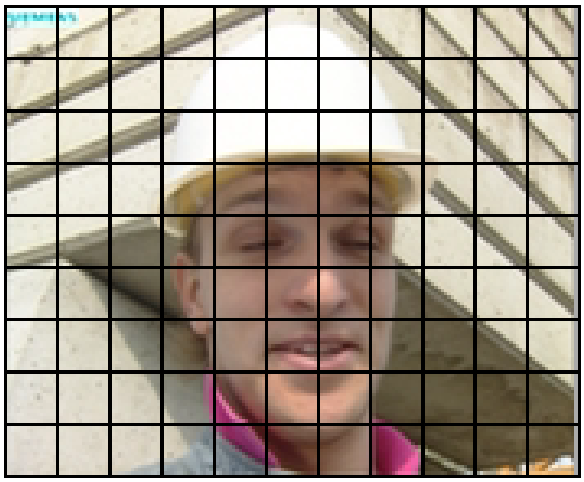


Figure 2.3: Frame Broken Down Into Macroblocks

Slice

A slice is formed from the combination of interrelated macroblocks within a frame. Slices may range in size from a single macroblock to the maximum number of macroblocks contained in a frame. Slices are formed such that inter-slice dependencies are minimized to limit error propagation across slice boundaries. H.264 defines five different slice types: I-Slice, P-Slice, B-Slice, SP-Slice and SI-Slice. Intra-Slices or I-Slices are composed of macroblocks that are predicted from data contained within the current frame. Predicted-Slices or P-Slices contain macroblocks that are predicted using the current frame and one previously encoded reference frame. Bi-Predictive-Slices or B-Slices are similar to P-Slices as they contain macroblocks predicted using current frame data and previously encoded frame data. However, B-Slices can contain macroblocks predicted using a combination of previously encoded frames whereas P-Slices can only use one previously encoded frame. Switching-P-Slices or SP-Slices and Switching-I-Slices or SI-Slices are used to transition between bit streams. SP and SI slices are used only in the Extended Profile and require specialized quantization [16]. Note that the Baseline Profile uses only I and P Slices.

The Baseline and Extended Profiles each support Arbitrary Slice Order (ASO) and redundant slices. ASO allows slices to be defined in non-raster

scan order such that enclosed macroblocks need not be adjacent. By providing non-raster scan order decoding, ASO allows for macroblocks within a slice to be dispersed throughout the frame such that a damaged slice will not omit continuous sections of a frame. In the event that an original slice is damaged when decoded, redundant slices allow for the inclusion of backup slices which may be decoded to take the place of the damaged slice.

Color Space

Modern devices conventionally use the Red, Green, Blue (RGB) color space to display images. Each pixel in RGB is broken down into three different values representing the proportion of red, green, and blue contained within the pixel. This color space is appropriate to provide the illusion of natural color. An example of a frame broken down into RGB components is shown in Figure 2.4.

(a) R (b) G (c) B

Figure 2.4: Frame Broken Down into RGB Components

However, the process of encoding pixels is not limited by the constraints of directly displaying pixels. H.264 uses the Luminance, Blue Chrominance, Red Chrominance (YCbCr) color space to represent each pixel during the encoding/decoding process. YCbCr allows greater flexibility in video encoding by breaking down images into luminance or brightness and chrominance or color difference. Luminance can be represented by a single value referred to as Y computed from a weighted sum of the RGB colors as shown in eq. (2.1) [15].

$$Y = k_r R + k_g G + k_b B \tag{2.1}$$

Chrominance is split into three components; one for each of the three additive primary colors of light consisting of red, green and blue each represented as a single value and referred to as Cr, Cg, and Cb respectively. Each chrominance component is determined from the difference of luminance and component color as shown in eqs. (2.2) to (2.4) [15].

$$Cb = B - Y \tag{2.2}$$
$$Cr = R - Y \tag{2.3}$$
$$Cg = G - Y \tag{2.4}$$

Observe that the equations for luminance and chrominance are not linearly independent. By storing Y and any two of the three chrominance components, the third chrominance component can be reconstructed. Thus YCbCr only stores blue and red chrominance values as green values can be reconstructed. An example of a frame broken down into YCbCr components is shown in Figure 2.5.

(a) Y (b) Cb (c) Cr

Figure 2.5: Frame Broken Down into YCbCr Components

Since videos are commonly captured and displayed in RGB, conversion between the RGB and YCbCr color spaces is necessary for encoding/decoding with YCbCr. Equations (2.5) to (2.7) show the conversion

process from RGB to YCbCr derived from eqs. (2.1) to (2.4) [15]. Likewise, eqs. (2.8) to (2.10) show the conversion process from YCbCr to RGB [15].

$$Y = k_r R + (1 - k_b - k_r)G + k_b B \tag{2.5}$$

$$Cb = \frac{0.5}{1 - k_b}(B - Y) \tag{2.6}$$

$$Cr = \frac{0.5}{1 - k_r}(R - Y) \tag{2.7}$$

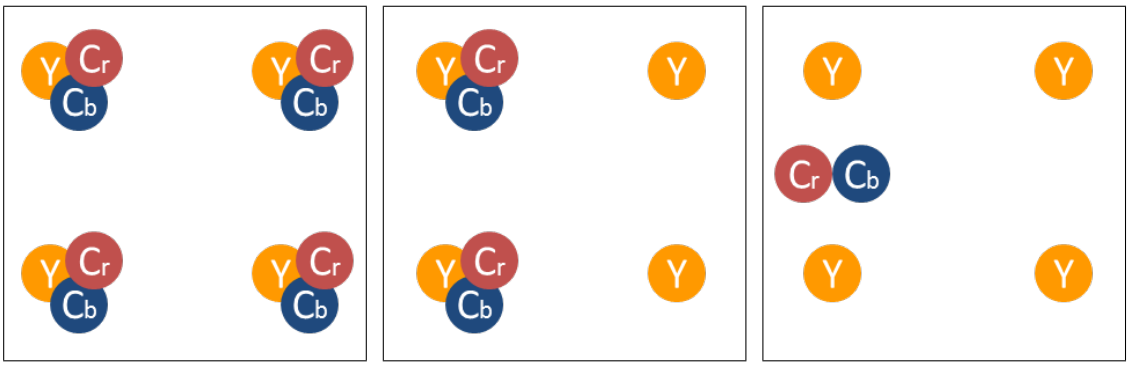
$$R = Y + \frac{1 - k_r}{0.5}Cr \tag{2.8}$$

$$G = Y - \frac{2k_b(1 - k_b)}{1 - k_b - k_r}Cb - \frac{2k_r(1 - k_r)}{1 - k_b - k_r}Cr \tag{2.9}$$

$$B = Y + \frac{1 - k_b}{0.5}Cb \tag{2.10}$$

The aforementioned color space conversion equations for RGB and YCbCr demonstrate the ability to represent a color in either space. Both RGB and YCbCr require storage of three values and thus neither color space would appear to be more advantageous for compressed storage than the other. However, the primary advantage of separating luminance and chrominance in YCbCr is that each can be stored at different resolutions. Since the Human Visual System (HVS) is more sensitive to luminance than chrominance [15], chrominance can be stored at a lower resolution than luminance without sacrificing perceived visual quality. This concept of utilizing fewer values than sampled is known as subsampling. H.264 supports three different sampling formats consisting of 4:4:4, 4:2:2, and 4:2:0. 4:4:4 does not incorporate any subsampling and thus stores Y, Cb, and Cr for each pixel. 4:2:2 subsamples chrominance values such that Cb and Cr values are only stored for every other horizontal pixel while Y values are stored for every pixel. 4:2:0 subsamples chrominance values such that Cb and Cr values are only

stored for every other horizontal and every other vertical pixel while Y values are stored for every pixel. A visual representation of the 4:4:4, 4:2:2, and 4:2:0 sampling formats is shown in Figure 2.6. While the H.264 standard supports each of the three previously mentioned sampling formats, the Baseline Profile only supports the 4:2:0 format.



(a) 4:4:4 (b) 4:2:2 (c) 4:2:0

Figure 2.6: YCbCr Sampling Formats

2.3.2 Prediction

Once the current frame has been broken down into macroblocks and grouped into slices, a predicted form of each macroblock is determined. H.264 breaks prediction down into two methods known as intra prediction and inter prediction (motion estimation). Intra prediction uses data from the current frame to form a prediction for the current block while motion estimation uses data from previously encoded frames to predict the current block. Predicted macroblocks are then subtracted from the actual macroblocks in order to form residuals. Residuals are beneficial as they minimize the amount of information required to reconstruct a macroblock and thus lead to higher compression ratios. By adding a predicted macroblock together with the corresponding residual, the macroblock can be fully reconstructed.

Rate-Distortion Optimization

Each prediction method specializes in identifying a specific movement pattern for a macroblock. Determining which prediction method is best suited

for the current macroblock requires an evaluation procedure to assess which method yields a block with minimal encoding cost. Rate-Distortion Optimization (RDO) balances the trade-off of image distortion versus the data cost of encoding a block.

Image distortion is evaluated by comparing the predicted block with the current block and attempting to minimize the resulting residual. To quantify which method minimizes the resulting residual, the Sum of Absolute Errors (SAE) is determined for each pixel in the predicted block in comparison with the corresponding pixel of the actual block. The equation for SAE is shown in eq. (2.11) where a is the actual value, p represents the predicted value, and n is the total number of values within the current block.

$$SAE = \sum_{i=0}^{n-1} |a_i - p_i| \tag{2.11}$$

A technique known as discretized Lagrangian optimization can be used for RDO in video coding to minimize the trade-off of SAE and cost of encoding the block. Equation (2.12) shows the Lagrangian cost definition for Lagrangian optimization where d is the distortion value evaluated from a metric such as SAE, λ is a Lagrangian multiplier, r is the rate or bit cost of encoding the block, and J is the resulting Lagrangian cost. The Lagrangian multiplier can be used to shift the resulting cost to minimize distortion when $\lambda = 0$ or to minimize the encoding cost when λ is large. Intermediate values of λ can be used to balance the distortion and encoding costs [13].

$$J = d + \lambda r \tag{2.12}$$

Intra Prediction

Intra prediction uses previously encoded macroblocks within the same frame to determine a predicted form of the current macroblock. To accomplish this, H.264 defines a total of thirteen intra prediction methods for the luminance component and four intra prediction methods for each of the chrominance components within a macroblock.

The luminance component of a macroblock can be processed for prediction as sixteen 4x4 blocks or left intact as a 16x16 block. If processed as 4x4 blocks, nine different intra prediction methods are available. These methods are described in Table 2.1 and illustrated in Figure 2.7. The prediction method which yields the smallest encoding cost is then selected to proceed to the next stage.

Mode	Method Name	Description
0	Vertical	The upper samples A, B, C, D are extrapolated vertically.
1	Horizontal	The left samples I, J, K, L are extrapolated horizontally.
2	DC	All samples are predicted by the mean of Samples A ... D and I ... L.
3	Diagonal Down-Left	The samples are interpolated at a 45° angle between lower-left and upper-right.
4	Diagonal Down-Right	The samples are extrapolated at a 45° angle down and to the right.
5	Vertical-Right	Extrapolation at an angle of approximately 26.6° to the left of vertical (width/height = 1/2).
6	Horizontal-Down	Extrapolation at an angle of approximately 26.6° below horizontal.
7	Vertical-Left	Extrapolation (or interpolation) at an angle of approximately 26.6° to the right of vertical.
8	Horizontal-Up	Interpolation at an angle of approximately 26.6° above horizontal.

Table 2.1: Intra Prediction Methods for 4x4 Luminance Blocks [15]

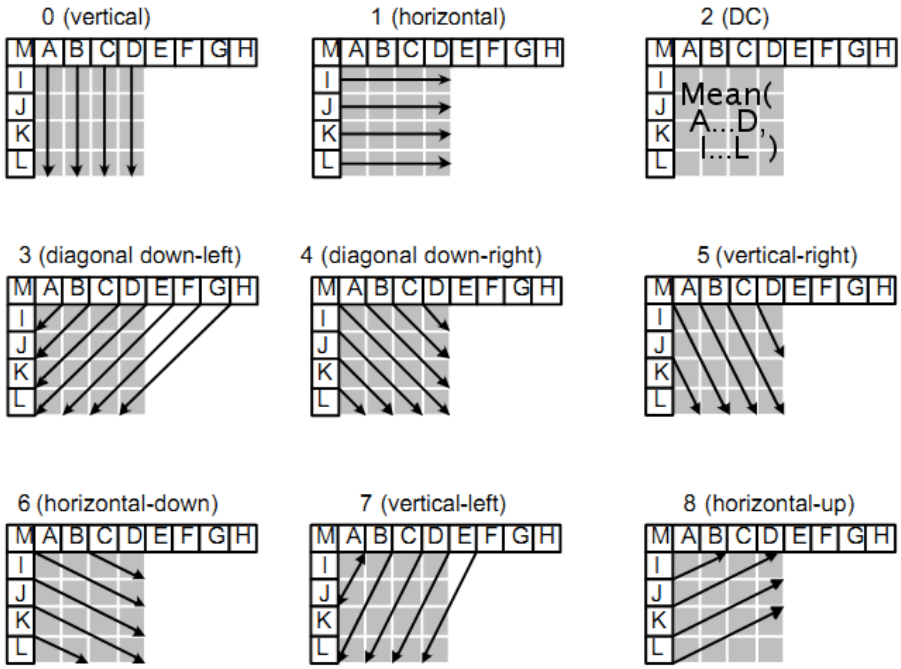


Figure 2.7: Intra Prediction Methods for 4x4 Luminance Blocks [15]

When luminance is predicted as a 16x16 block, four different intra prediction methods are available. These same methods are available for 8x8 chrominance blocks. Table 2.2 describes the luminance and chrominance methods while Figure 2.8 illustrates each mode.

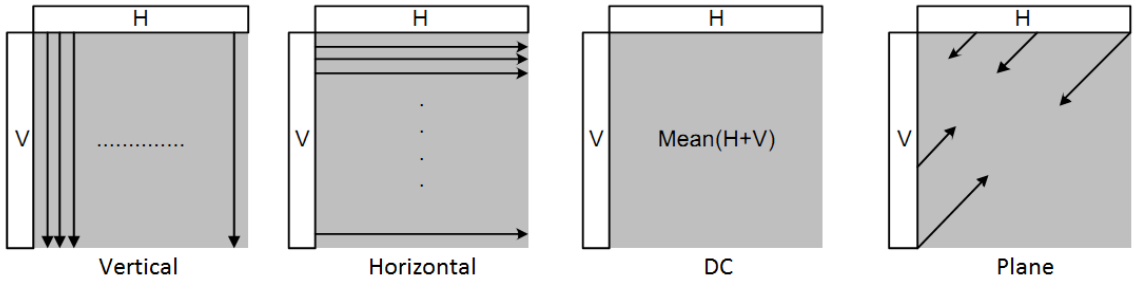


Figure 2.8: Intra Prediction Methods for 16x16 Luminance Blocks and 8x8 Chrominance Blocks [15]

Luminance Mode	Chrominance Mode	Method Name	Description
0	2	Vertical	Extrapolation from upper samples (H).
1	1	Horizontal	Extrapolation from left samples (V).
2	0	DC	Mean of upper and left-hand samples (H + V).
3	3	Plane	A linear 'plane' function is fitted to the upper and left-hand samples H and V. This works well in areas of smoothly-varying luminance.

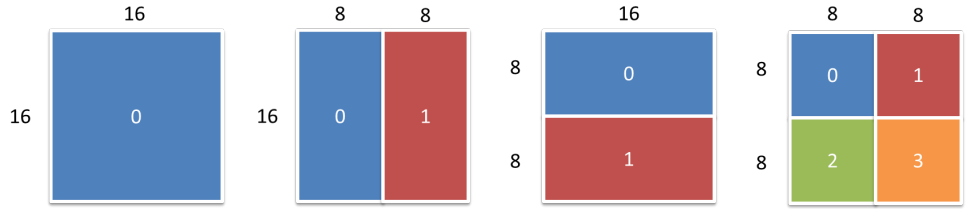
Table 2.2: Intra Prediction Methods for 16x16 Luminance Blocks and 8x8 Chrominance Blocks [15]

Motion Estimation

Motion estimation or inter prediction uses previously encoded frames to predict blocks in the current frame and thus minimize residuals for the current frame. Pixels in a given frame often correlate strongly with pixels in neighboring frames. To take advantage of this correlation, motion estimation computes motion vectors which describe the movement of a block in a previous frame to the current.

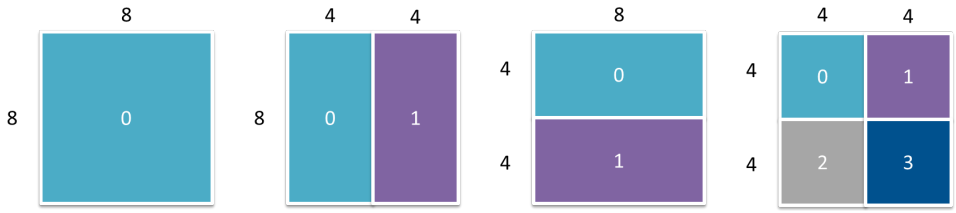
Since objects in a frame may vary in size and likely require higher resolution than 16x16 pixel macroblocks, H.264 defines macroblock partitions and sub-macroblock partitions for use in motion estimation. Each 16x16 macroblock can be partitioned into two 8x16 blocks, two 16x8 blocks, four 8x8 blocks or left intact as a 16x16 block. In the event that 8x8 block partitions are formed, each of these partitions can be broken down further into two 4x8 blocks, two 8x4 blocks, four 4x4 blocks or left intact as 8x8 blocks [15]. Blocks smaller than 8x8 are considered sub-macroblock partitions as they divide macroblock partitions further. Figure 2.9 shows the four possible macroblock partitions while Figure 2.10 shows the four possible sub-macroblock partitions. Note that the aforementioned macroblock and sub-macroblock partitions correspond to the luminance component of a macroblock. The chrominance components of a macroblock use half the width

and height of the luminance block partitions in 4:2:0 encoding. Thus, the smallest chrominance macroblock partition and sub-macroblock partition is 4x4 and 2x2 respectively.



(a) 16x16 (b) 8x16 (c) 16x16 (d) 8x8

Figure 2.9: Macroblock Partition Formats



(a) 8x8 (b) 4x8 (c) 8x4 (d) 4x4

Figure 2.10: Sub-Macroblock Partition Formats

The motion estimation stage is often considered the most computationally intensive portion of the video encoding process [18]. This is due to the number of blocks and sub-block configurations throughout multiple frames which must be searched to minimize resulting residuals. H.264 adds to this complexity of motion estimation by providing fractional motion vector support. Considering that objects within previous frames likely do not move in integral pixel increments between frames, H.264 provides support for fractional motion vectors which can interpolate objects moving to quarter pixel locations. Fractional motion vectors require additional computation and additional space to store the fractional portion of the motion vector. However, higher compression ratios can be achieved as residuals should decrease and result in less data for encoding.

Due to the complexity of motion estimation, various algorithms have been introduced to balance the trade-off of computational complexity versus

compression ratio. Full Search [15], Fast Full Search [1], Unsymmetrical-cross and Multi-Hexagon-grid (UMHex) Search [5], Simplified UMHex Search [25] and Enhanced Predictive Zonal Search (EPZS) [18] comprise five mainstream motion estimation algorithms for H.264. In addition, this thesis involves the implementation of a hardware accelerator based on the Hardware-Oriented Modified Diamond Search (HMDS) motion estimation algorithm presented in [10].

Full Search

Full Search motion estimation examines every possible block combination within a given search window to find the optimal solution. As with intra prediction, inter prediction requires a method such as SAE to evaluate the benefits of one block versus another. Full Search evaluates the error for each block permutation. This exhaustive nature of Full Search produces high compression ratios at the cost of increased computational complexity. The computational complexity of Full Search results in longer encoding times consuming up to 80% of the encoding process [1].

Fast Full Search

Fast Full Search improves upon Full Search by reducing the number of error evaluations. Rather than determining the error for every block combination, error calculation is only directly performed for 4x4 blocks; the smallest inter prediction blocks. By obtaining error values for each of the 4x4 blocks, the error for the remaining block sizes: 8x4, 4x8, 8x8, 16x8, 8x16, and 16x16 can be estimated by accumulating the error of the 4x4 blocks by which they are composed.

UMHex Search

UMHex Search reduces the computational complexity of the Full Search methods by searching only a subset of the search window. The search uses a four step process to determine a motion vector. To begin, UMHex calculates a median predictor in which the motion vector for the current block is estimated from the median motion vector of the left, top, and top-right blocks that have previously been determined. Figure 2.11 shows the location of the current block in relation to previously encoded blocks while eq. (2.13) expresses the mathematical computation required for the median predictor [1].

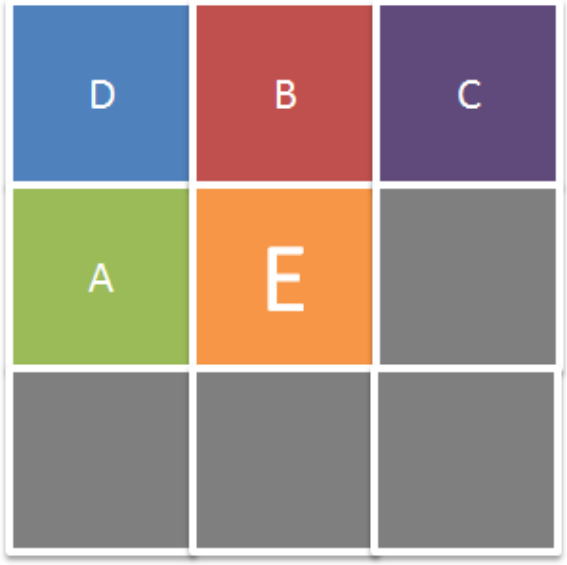


Figure 2.11: Current Block, E, with respect to Previously Encoded Blocks [1]

$$MV_{E,median} = median(MV_A, MV_B, MV_C) \tag{2.13}$$

Using the median predictor as a starting point, UMHex uses an unsymmetrical-cross search to evaluate surrounding points. The cross is usually weighted more heavily in the horizontal direction as videos tend to contain predominantly horizontal movement rather than vertical [1]. In addition, the cross may skip blocks horizontally and vertically to minimize the number of blocks

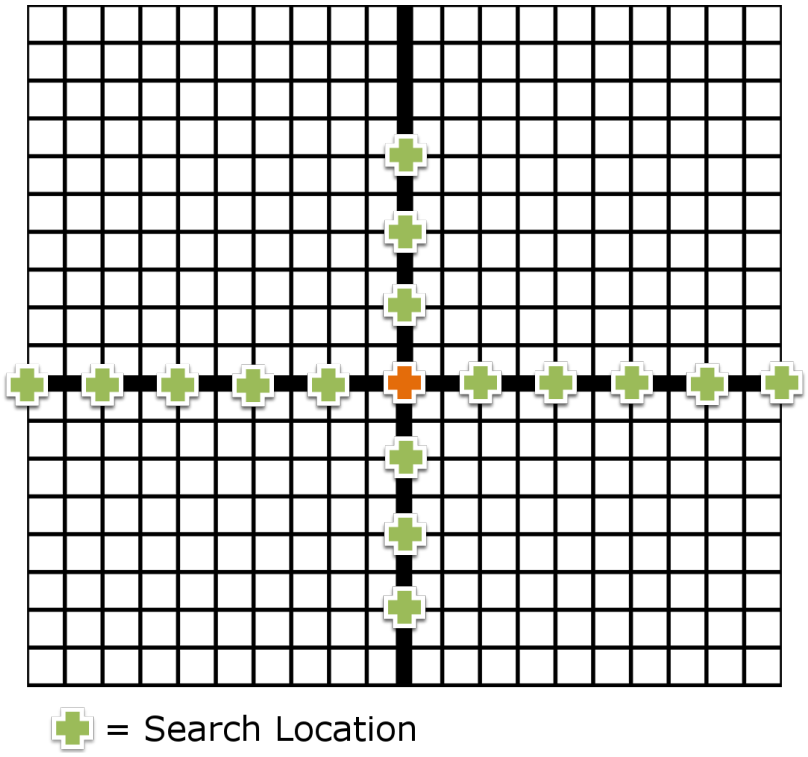


Figure 2.12: Example of Unsymmetrical-Cross Search Pattern

examined. An example of an unsymmetrical-cross search pattern is shown in Figure 2.12.

Third, a full search encompassing a 5x5 region centered at the best motion vector identified through the unsymmetrical-cross search is performed. In addition, a sixteen point hexagon pattern search centered at the best motion vector from the cross search is performed to account for motions exceeding the +/- 2 full search region. This sixteen point hexagon pattern can be scaled to search locations in hexagon rings at distances further from the current center location. Figure 2.13 depicts a single sixteen point hexagon pattern.

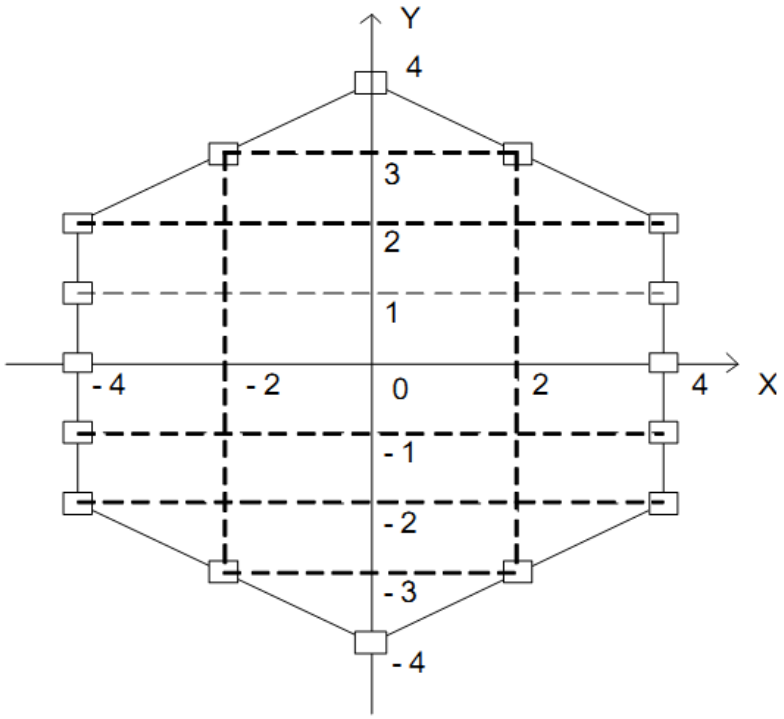
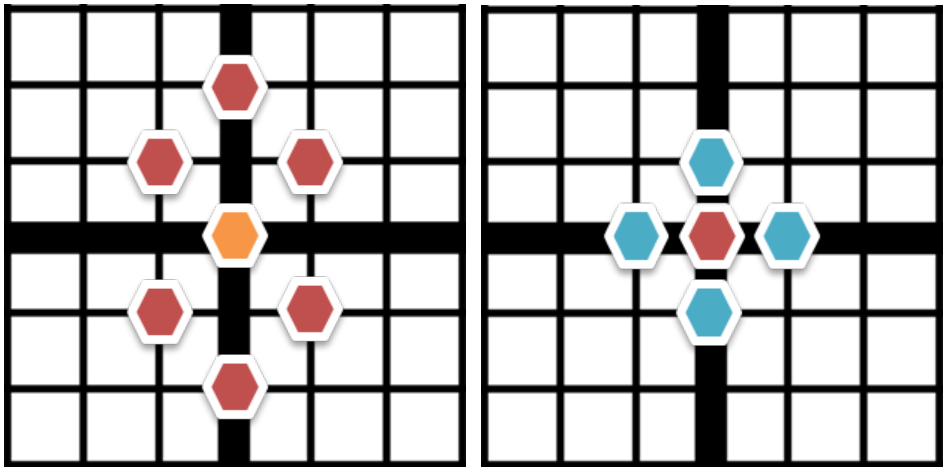


Figure 2.13: Sixteen Point Hexagon Search Pattern [1]

Finally, the best motion vector resulting from the combined 5x5 full search and sixteen point hexagon searches is used as a basis for a hexagon based search. This search begins using the large hexagon pattern shown in Figure 2.14a centered at the current best motion vector. If after searching each of the six locations in the large hexagon pattern the best motion vector remains the center location, the small hexagon pattern shown in Figure 2.14b is used, otherwise, the large hexagon pattern search is applied again centered at the new motion vector. The best motion vector resulting from the small hexagon pattern search is then declared the best motion vector for the UMHex Search [5]. Figure 2.15 gives an example of what the entire UMHex Search process may look like from start to finish. Note that the center location (0,0) in Figure 2.15 represents the result of the initial prediction phase, also known as step 1.



(a) Large Hexagon Pattern (b) Small Hexagon Pattern

Figure 2.14: Hexagon Based Search Patterns

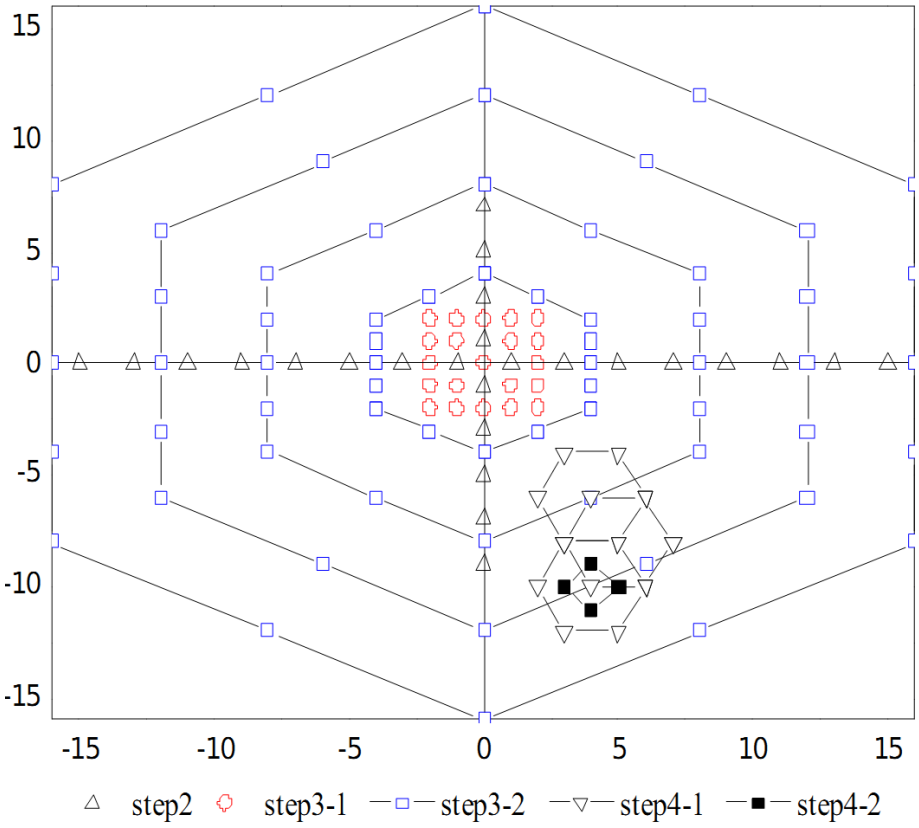


Figure 2.15: UMHex Search Example [1]

Simplified UMHex Search

Simplified UMHex Search improves upon the standard UMHex Search by adding additional initial prediction motion vectors and the ability to terminate the search early if certain criterion are met [25]. In addition to the initial median predictor used in the first stage of UMHex Search, Simplified UMHex Search adds UpLayer prediction, and Last Reference Frame (LRF) prediction to the list of possible starting motion vector values. UpLayer prediction takes advantage of the fact that the same area of a frame will be analyzed with block sizes ranging from 16x16 to 4x4 by using the motion vector previously predicted for large blocks as a starting point for small blocks. For instance, an 8x16 block would use the predicted motion vector from the corresponding 16x16 block as a starting point. LRF prediction uses a scaled version of the motion vector predicted from a previously encoded reference frame as an initial motion vector. The equation for scaling the motion vector during LRF prediction is shown in eq. (2.14) where t is the current frame number and t' is the reference frame number [2]. Lastly, Simplified UMHex Search adds early termination criteria such that when the variance of the current motion vector with respect to the previously predicted motion vector is less than a modulated threshold, the search skips to the final stage, hexagon based search, of the UMHex Search process to determine a final motion vector. A flowchart which demonstrates the overall Simplified UMHex Search is shown in Figure 2.16.

$$MV_{predictor,LRF} = MV_{LRF} * \frac{t - t'}{t - t' - 1} \tag{2.14}$$

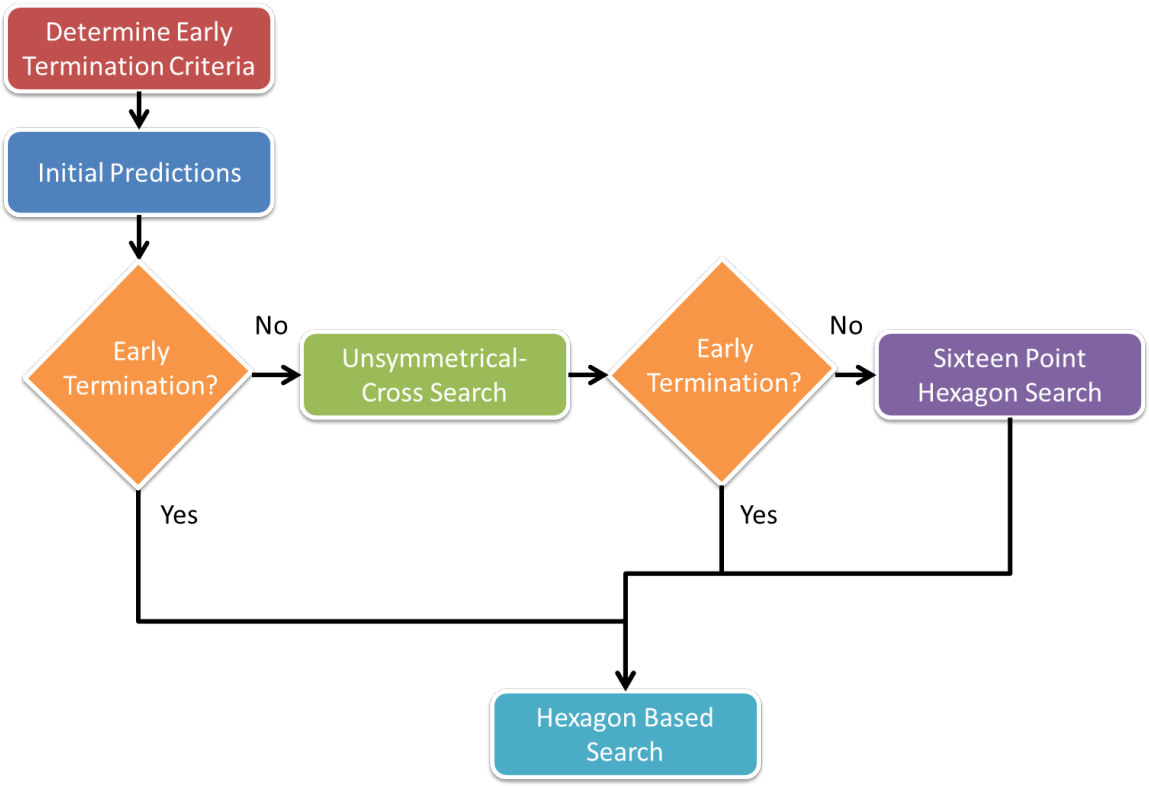


Figure 2.16: Simplified UMHex Flowchart [26]

EPZS

EPZS, like the UMHex Searches attempts to limit the number of blocks analyzed by starting with predicted motion vectors. However, unlike the UMHex Searches which start with a small set of initial predictors and then branch out with a broad set of search locations, EPZS focuses on a large set of initial predictors before moving on to a relatively simplistic search pattern. EPZS begins with a three stage initial predictor phase. The first stage investigates the median predictor and checks the error against a threshold. If the threshold criteria is met, the algorithm terminates using the median predictor. If not, the next stage examines the (0,0) motion vector in addition to four motion vectors from the current frame consisting of the left block, top-left block, top block, and top-right block or A, D, B, C respectively according to Figure 2.11. If the best motion vector within this stage does not meet the threshold, the search continues on to the final prediction stage. The

final stage analyzes motion vectors from the previous frame corresponding to the current block location, the block to the left, the block above, the block to the right, and the block below or F, G, H, I, and J of Figure 2.17 [19]. If the best motion vector from the final initial predictor stage does not meet the early termination criteria, a pattern search is performed centered at the current best motion vector. EPZS offers three different search patterns. In order of increasing computational intensity, these patterns consist of small diamond, square and extended and are shown in Figure 2.18. EPZS allows the use of any of the three aforementioned patterns to provide flexibility in the encoding process. For each pattern, the search examines the specified blocks and selects the best motion vector. The search is then repeated with the pattern shifted such that the center of the pattern is the previous best motion vector. Similar to UMHex, this process repeats until the center motion vector remains the best motion vector at which point the EPZS algorithm concludes [18].

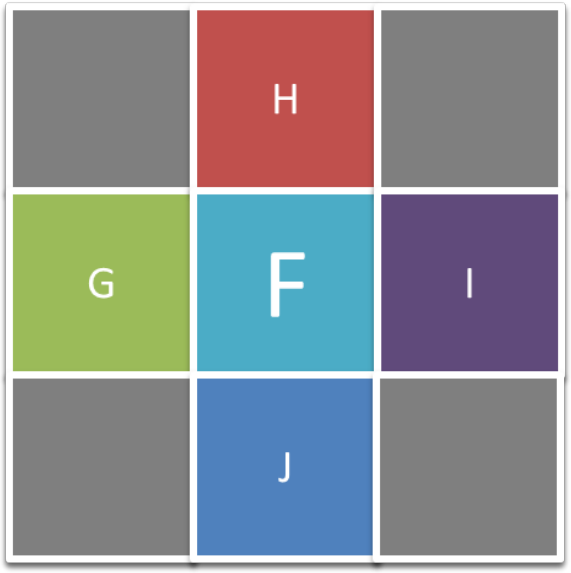
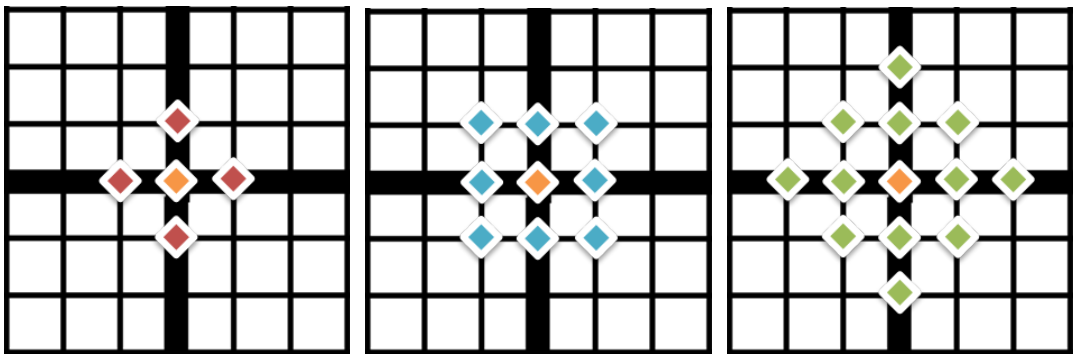


Figure 2.17: Previous Reference Frame with Block F corresponding to Block E of the Current Frame as shown in Figure 2.11



(a) Small Diamond Pattern (b) Square Pattern (c) Extended Pattern

Figure 2.18: EPZS Patterns

HMDS

HMDS uses search techniques similar to that of UMHex and EPZS adjusted for low bit rate, low-power hardware devices. Due to the complexity of determining initial motion vector predictors, HMDS always begins with the (0,0) motion vector. Using a simple three step procedure, HMDS proceeds to examine the search pattern demonstrated in Figure 2.19. Note that the search locations close to the center of Figure 2.19 are to scale whereas the four points at the edge are not. The two points on the negative axes are at distance N from the search origin while the positive axis points are at distance N-1 where N is the predefined search region. The best motion vector from the first search pattern is then used as the center of the extended search pattern from EPZS as shown in Figure 2.18c. The third step uses the extended pattern centered at the best motion vector from the previous step to determine the final motion vector. The algorithm examines a constant total of 41 locations per run [10].

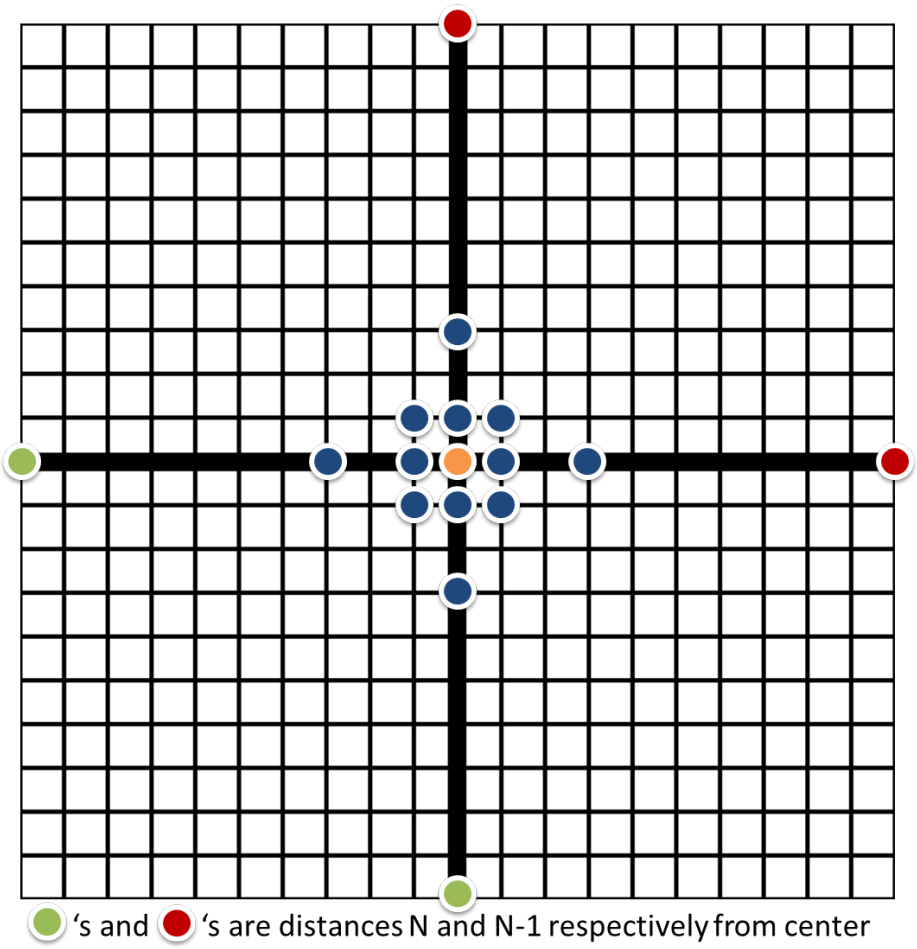


Figure 2.19: First HMDS Search Pattern [10]

2.3.3 Core Coding

Following intra/inter prediction, the "Core Coding" phase transforms and quantizes blocks in preparation for entropy encoding. For each macroblock residual formed in prediction, a "core" transform is performed on the 4x4 blocks contained within. Note that the luminance component of a macroblock contains sixteen 4x4 blocks, while the chrominance component contains four 4x4 blocks. After the transform stage, further transformation and quantization is split into three categories: 16x16 intra prediction luminance blocks, chrominance blocks, and all remaining blocks.

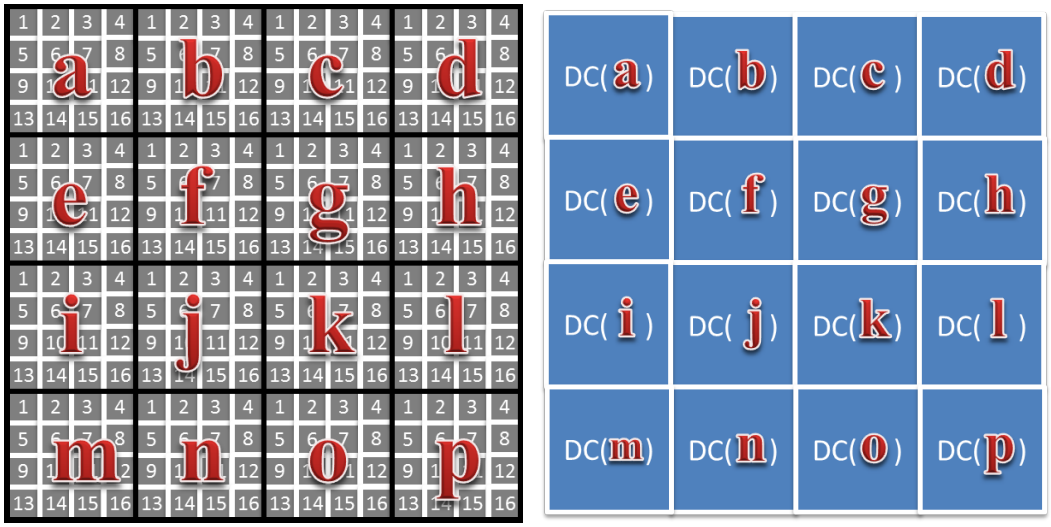
Transform

The primary goal of the transform process is to shift the macroblock residuals created during prediction from the spatial to the frequency domain. Use of the frequency domain facilitates determination of the average or DC component of a block and is beneficial for quantization. H.264 uses a specialized Discrete Cosine Transform (DCT) referred to as the "core" transform to accomplish this modification. Unlike a conventional DCT, the core transform utilizes purely integer operations and can be implemented with addition and shift operations. The equation for the core transform is shown in eqs. (2.15) and (2.16) where $\otimes$ indicates element by element multiplication, X represents the 4x4 residual block input, L, M, and N represent the constants $\frac{1}{4}$, $\frac{114}{721}$, and $\frac{1}{10}$ respectively and Y represents the result of the transform [15].

$$Y = (CXC^T) \otimes E \tag{2.15}$$

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} \\ X \\ \\ \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} L & M & L & M \\ M & N & M & N \\ L & M & L & M \\ M & N & M & N \end{bmatrix} \tag{2.16}$$

After the core transform, 16x16 intra prediction luminance and chrominance blocks are transformed further while the remaining blocks move directly to quantization. For 16x16 intra prediction luminance blocks, a new 4x4 matrix is created composed of the averages of each 4x4 block as shown in Figure 2.20.



(a) 16x16 Intra Prediction Luminance Block (b) Average 4x4 Intra Prediction Luminance Block

Figure 2.20: Formation of 4x4 DC Luminance Block from 16x16 Luminance Block

Next, a 4x4 Hadamard transform is applied to the DC matrix to fully prepare the block for quantization according to the equation shown in eq. (2.17) where W represents the input 4x4 average matrix as shown in Figure 2.20b, and Y represents the output of the Hadamard transform.

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & W & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) /2 \qquad (2.17)$$

For each chrominance block, a similar transform is performed on a 2x2 average matrix created from the 8x8 transformed chrominance block to prepare the block for quantization. The equation for this transform is shown in eq. (2.18) where W represents the input 2x2 average matrix and Y represents the output of the transform.

$$Y = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} & W & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (2.18)$$

Quantization

The quantization stage seeks to reduce the range of transformed residual values such that fewer bits are required for storage. Quantization in H.264 is configured through the use of two Quantization Parameters (QPs); one for luminance and the other for chrominance. Each QP must be within the range 0 - 51 for which smaller values represent higher bit rates and greater quality while larger values indicate lower bit rates and lesser quality. QStep values are then determined from the specified QP values according to Table 2.3 [15]. Note that QStep values double every six increments of QP. In addition, an intermediate variable, qbits, is derived from QP according to eq. (2.19) [15].

QP	0	1	2	3	4
QStep	*0.625*	*0.6875*	*0.8125*	*0.875*	*1*
QP	5	6	7	8	9
QStep	*1.125*	*1.25*	*1.375*	*1.625*	*1.75*
QP	10	11	12	18	24
QStep	*2*	*2.25*	*2.5*	*5*	*10*
QP	30	36	42	48	51
QStep	*20*	*40*	*80*	*160*	*224*

Table 2.3: QP to QStep Mapping [15]

$$qbits = 15 + floor(QP/6) \tag{2.19}$$

For 16x16 intra prediction luminance blocks, each value of the transformed 4x4 average blocks is quantized according to the eq. (2.20) [15] where $Y_{(i,j)}$ refers to the current value within the transformed 4x4 average luminance block, $Z_{(i,j)}$ refers to the resulting quantized value and $QStep$ and $qbits$ are defined as above.

$$\begin{aligned} |Z_{(i,j)}| &= (|Y_{(i,j)}| * \frac{1}{4 * QStep} + \frac{2^{qbits+1}}{3} >> (qbits + 1) \\ sign(Z_{(i,j)}) &= sign(Y_{(i,j)}) \end{aligned} \tag{2.20}$$

Chrominance average 2x2 transformed blocks are quantized according to eq. (2.21) [15] where $Y_{(i,j)}$ refers to the current value within the transformed 2x2 average chrominance block, $Z_{(i,j)}$ refers to the resulting quantized value, QStep and qbits are defined as above, and a is 1 for intra prediction blocks and 2 for inter prediction blocks.

$$|Z_{(i,j)}| = (|Y_{(i,j)}| * \frac{1}{4 * QStep} + \frac{2^{qbits+1}}{a * 3} >> (qbits + 1)$$
$$sign(Z_{(i,j)}) = sign(Y_{(i,j)}) \tag{2.21}$$

For all remaining blocks (non-16x16 intra prediction and all inter prediction luminance blocks) quantization is performed according to eq. (2.22) [15] where $Y_{(i,j)}$ refers to the current value within the transformed block, $Z_{(i,j)}$ refers to the resulting quantized value, QStep, qbits, and a are defined as above, and PF is a position factor. The position factor is determined by the location within the 4x4 block as defined in Figure 2.21 [15].

$$|Z_{(i,j)}| = (|Y_{(i,j)}| * \frac{PF}{QStep} + \frac{2^{qbits}}{a * 3} >> qbits$$
$$sign(Z_{(i,j)}) = sign(Y_{(i,j)}) \tag{2.22}$$

$\frac{1}{4}$	$\frac{114}{721}$	$\frac{1}{4}$	$\frac{114}{721}$
$\frac{114}{721}$	$\frac{1}{10}$	$\frac{114}{721}$	$\frac{1}{10}$
$\frac{1}{4}$	$\frac{114}{721}$	$\frac{1}{4}$	$\frac{114}{721}$
$\frac{114}{721}$	$\frac{1}{10}$	$\frac{114}{721}$	$\frac{1}{10}$

Figure 2.21: Position Factor Mapping

2.3.4 Entropy Coding

Entropy coding describes the stage in which all information needed for decoding is gathered into a bit string format which maximizes compression. Essential information required for decoding includes: slice syntax elements, macroblock prediction types, QPs, frame indices, motion vectors, and residual data [15]. Context-Adaptive Binary Arithmetic Coding (CABAC) or Variable Length Coding (VLC) are two modes available for H.264 entropy coding. CABAC provides high compression at the cost of increased computational complexity and is only supported in the Main Profile. VLC is less computationally complex and is divided into two primary techniques: Context-Adpative Variable-Length Coding (CAVLC) and Exponential-Golomb coding. CAVLC uses a series of methods and mappings to further reduce data redundancy and thus increase compression while coding the essential video components. Exponential-Golomb is a simple method for mapping numerical values to variable length bit strings. Data coded using Exponential-Golomb is mapped according to Table 2.4 before concatenation to the current bit string.

Data	**Code**
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	
N	[$floor(log_2[N+1])$ 0's][1][$N+1-2^{floor(log_2[N+1])}$]

Table 2.4: Exponential-Golomb Data Mapping [15]

2.3.5 Network Abstraction Layer

The Network Abstraction Layer (NAL) is responsible for preparing the coded data for storage or network transmission. It breaks down coded data

into NAL units which contain a header and Raw Byte Sequence Payload (RBSP). NAL units can then be sent in individual packets across a network or stored in a file for later use. The NAL provides a level of abstraction which facilitates distribution of the coded H.264 video.

2.4 Video Quality Metrics

Quantitative quality measurements can be broken down into two categories: objective and subjective. Determining a realistic objective quality metric is difficult due to the complexity of the HVS while determining a subjective quality metric proves problematic as opinions vary amongst the populace. When humans observe an image, focus is normally drawn to a specific point within an image such as a face. Areas outside of this focal point are given less consideration. Humans have been shown to determine quality based on the following factors: areas of interest such as people, location where the video is watched, previous video watching experience, display type, viewing conditions, quality of audio, and level of interaction the user has with the video [20]. With many of these factors varying based on the viewer, subjective quality measurements must be based on average feedback from a large group.

Due to the costs associated with gathering a large group of people to determine the quality of a video, objective quality metrics for which an algorithm can evaluate quality are commonly used. Peak Signal-to-Noise Ratio (PSNR) is one of the most widely used objective metrics as it is relatively simple to understand and compute [20]. PSNR represents the Mean Squared Error (MSE) error in logarithmic form. The equation for MSE is shown in eq. (2.23) where a represents the actual data value, c represents the coded data value, and n and m represent the block height and width. Note that increasing MSE values indicate loss of quality.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [a_{i,j} - c_{i,j}]^2 \tag{2.23}$$

The equation for PSNR is given in eq. (2.24) where n represents the number of bits used to store the block. Note that increasing PSNR values indicate gain in quality with the limit approaching infinity where the original image matches the coded image. Furthermore, the average PSNR for an entire video sequence is determined by averaging the PSNRs for each frame. In addition, PSNR can be computed separately for each of the three color components, Y, Cr, and Cb or an overall PSNR frame value can be computed by treating each component value as a separate pixel and dividing by an additional factor of three.

$$PSNR = 10 * log_{10}\frac{(2^n - 1)^2}{MSE} \tag{2.24}$$

Although PSNR can be used as a relative guideline for quality, the metric is not an adequate replacement for subjective quality metrics. PSNR examines pixels one at a time without regard to their relation to each other. In [20], the example shown in Figure 2.22 is provided to demonstrate that two images can have identical PSNR values in reference to the same image yet, given a subjective evaluation, Figure 2.22b would clearly be deemed of lower overall quality than Figure 2.22a.

2.5 H.264 JM Reference Encoder

In addition to publishing the H.264 standard, JVT produces an H.264 CODEC which adheres to the standard. This CODEC is provided as part of the JM Reference Software package. The software is written in C and constructed in such a way that the terminology of the standard closely matches that of the software. Furthermore, the software includes numerous statistics for evaluating the performance of the encoder. However, the JM Reference Software CODEC efficiency is substantially less than that of alternative H.264 CODECs. Thus, the JM Reference Software is primarily used for research rather than production efforts.

The JM Reference Software contains all of the aforementioned components with the exception of HMDS motion estimation.

(a) (b)

Figure 2.22: Identical PSNR Comparison [20]

2.6 Region of Interest Encoding

RoI encoding uses information about the foreground or which parts of the frame are of interest to the viewer to improve the quality and/or reliability of the encoded video. Specifically, regions of interest may be allotted higher bit rates to give the illusion that the entire frame is of better quality. In error prone environments, regions of interest may be redundantly encoded to increase the likelihood that portions of the frame which contain regions of interest will be properly decoded.

RoI encoding methods primarily modify the QP to adjust the encoded bit rate for the foreground and background sections of the frame to achieve a relatively higher perceived frame quality. The Baseline and Extended Profile capability of encoding redundant slices is utilized to encode duplicate slices for regions of interest.

2.7 Field Programmable Gate Arrays

FPGAs consist of a multitude of reconfigurable hardware resources. Through the use of Hardware Description Languages (HDLs), FPGA resources may be programmed in such a way that a working logical system is formed. HDLs such as Very-high-speed integrated circuit Hardware Description Language (VHDL) and Verilog simplify design by representing hardware at the functional block level rather than the gate level. Dependent on the technology, some FPGAs such as Static Random-Access Memory (SRAM) based FPGAs allow reprogramming while others such as fuse based FPGAs can only be programmed once. The ability to reprogram an FPGA facilitates rapid hardware prototyping which is not possible using traditional Application-Specific Integrated Circuit (ASIC) production methods. Full ASIC design is a lengthy process which requires multi-layer layouts to form and connect transistors in order to create a working system. Moreover, once an ASIC has been designed, it must be fabricated in silicon which is a costly and time consuming process resulting in a piece of hardware which cannot be modified.

Despite definitive advantages such as reconfigurability and ease of design, modern FPGAs cannot serve as complete replacements for ASICs. FPGAs achieve reconfigurability through large arrays of logic cells which are connected to each other through programmable switches. These switches and functional blocks require additional power and limit the performance of the system in comparison with ASICs. Thus, some companies have chosen a hybrid design approach in which designs are started and verified with FPGAs. Once a design has been proven, it is then converted to an ASIC layout through modern tools. The resulting layout can then be tweaked at the transistor level before being sent out for fabrication. This hybrid approach helps companies to reap the benefits of both FPGAs and ASICs.

Recent developments in FPGAs include the ability to reconfigure sections of the FPGA at runtime. Partial reconfiguration allows smaller FPGAs to hold designs which would be too large to fit on the FPGA at one time. In addition, partial reconfiguration can be leveraged to eliminate hardware resources which may drain power while not in use.

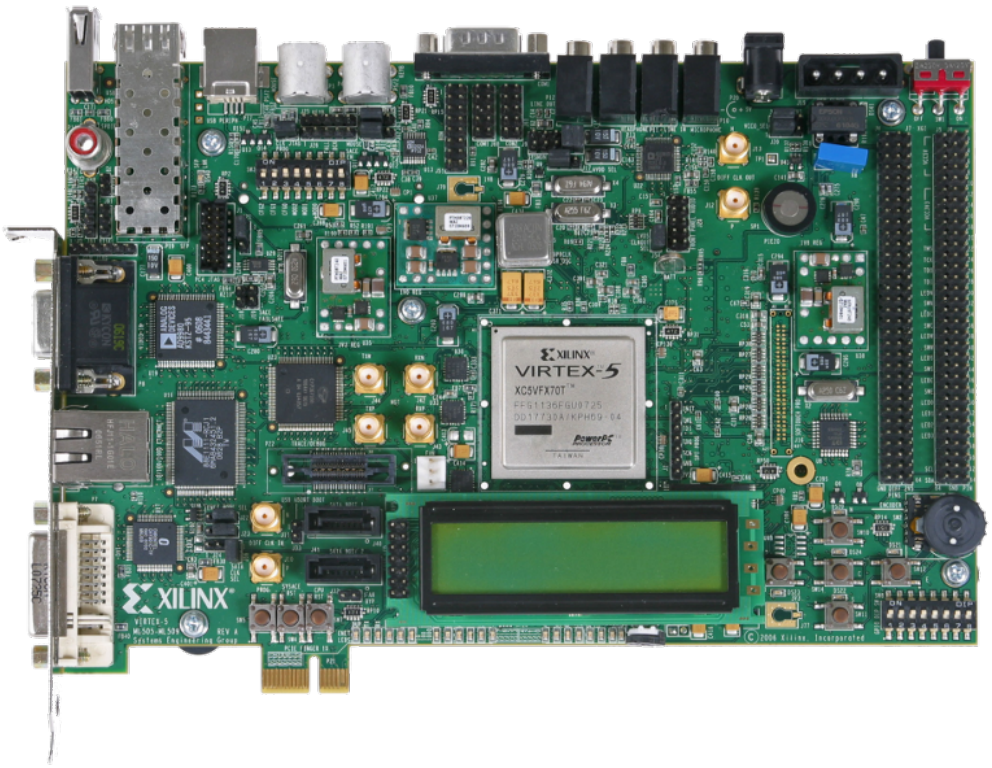
Despite the flexibility, performance and ease of design of hardware provided by FPGAs and HDLs, General Purpose Processors (GPPs) and software still maintain advantages such as fast sequential operation of instructions and shorter development times. In an attempt to gain the benefits of both hardware and software, FPGA manufacturers such as Xilinx and Altera have released FPGAs coupled with GPPs. FPGA GPPs are further broken down into two categories: hard core and soft core. An FPGA hard core GPP is one which does not require FPGA resources. Alternatively, a soft core GPP must be synthesized using FPGA resources.

2.7.1 Xilinx ML-507 Development Board

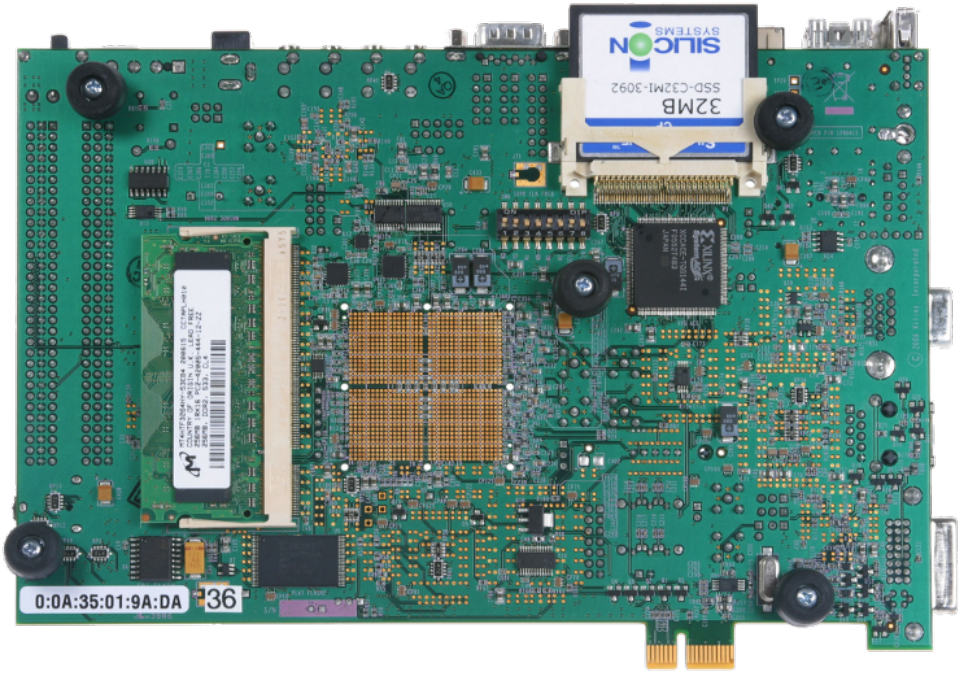
The Xilinx ML-507 Development board includes a Xilinx Virtex 5 XC5VFX70T with a PowerPC 440 processor built-in along with numerous I/O connections, 256 MB DDR2 memory, and 1 MB SRAM. Images of the board are shown in Figure 2.23.

Xilinx Virtex 5 XC5VFX70T

The Xilinx Virtex 5 XC5VFX70T FPGA is built on 65 nm technology and contains 6,080 resource blocks or what Xilinx refers to as Configurable Logic Blocks (CLBs). These CLBs are laid out in a 160x38 array connected by a switching matrix and altogether contain a total of 44,800 6-input Look-Up Tables (LUTs), 5,328 Kb of Block Random-Access Memory (BRAM), and 44,800 Flip-Flops [21]. Besides the standard FPGA resources, the XC5VFX70T supports on-board processor communication over Processor Local Buses (PLBs). In addition to the previously mentioned PowerPC 440 physically included on the chip, Xilinx provides the capability to program the FPGA with a proprietary soft processor core known as the MicroBlaze.



(a) Front



(b) Back

Figure 2.23: Xilinx ML-507 Development Board

Processor Local Bus

The XC5VFX70T contains 128-bit PLBs to support processor communication to FPGA resources. Three PLBs are dedicated to supporting the PowerPC processor for instruction and data cache reads and writes [22]. A fourth PLB is used to communicate with peripherals including SRAM, DDR2, Universal Asynchronous Receiver/Transmitters (UARTs), timers, push buttons, Light-Emitting Diodes (LEDs) and custom peripherals. Resources connected to the PLB must be established as masters or slaves. A master may initiate and respond to requests on the bus while a slave can only respond to requests. The PLB contains a Watchdog Timer to initiate timeout responses when necessary. Figure 2.24 shows a block diagram of the PLB.

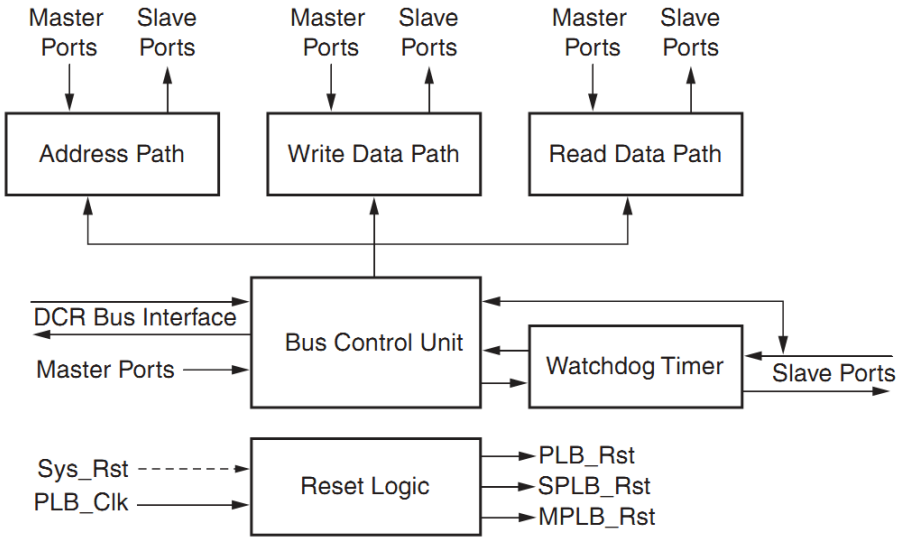


Figure 2.24: PLB Block Diagram [22]

PowerPC 440

The XC5VFX70T's included IBM PPC 440x5 32-bit Reduced Instruction Set Computer (RISC) processor core can run at a maximum clock speed of 400 MHz. It contains 32KB 64-way associative instruction and data caches, thirty-two 32-bit General Purpose Registers (GPRs), a 32-bit address bus, and a 64-bit timer. To gain floating point support, a Floating Point Unit (FPU) can be synthesized using FPGA resources. A block diagram of the included chip is shown in Figure 2.25.

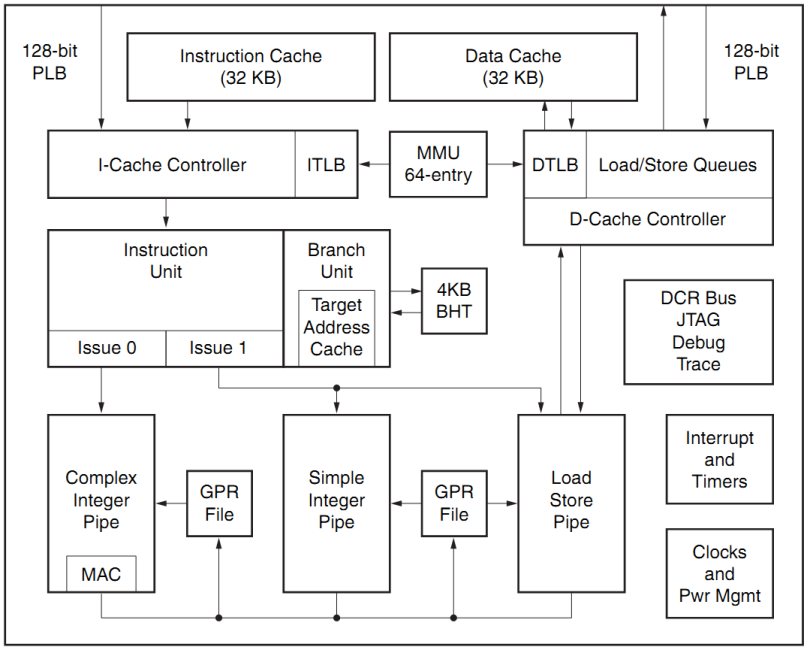


Figure 2.25: PowerPC 440 Block Diagram [23]

MicroBlaze

The XC5VFX70T can be programmed with Xilinx's 32-bit RISC soft core processor known as the MicroBlaze processor. The MicroBlaze on the XC5VFX70T can perform at a maximum clock speed of 125 MHz. It contains thirty-two 32-bit GPRs, a 32-bit address bus and contains configurable 1-way associative data and instruction caches with sizes up to 64KB each [24]. Like the PPC, a FPU for the MicroBlaze processor may be configured using additional FPGA resources. Since the MicroBlaze processor is

synthesized using FPGA elements, the use of the MicroBlaze reduces the number of FPGA resources available for custom logic. The final number of FPGA resources available for custom logic is dependent on the MicroBlaze configuration. Figure 2.26 shows a block diagram of the MicroBlaze processor.

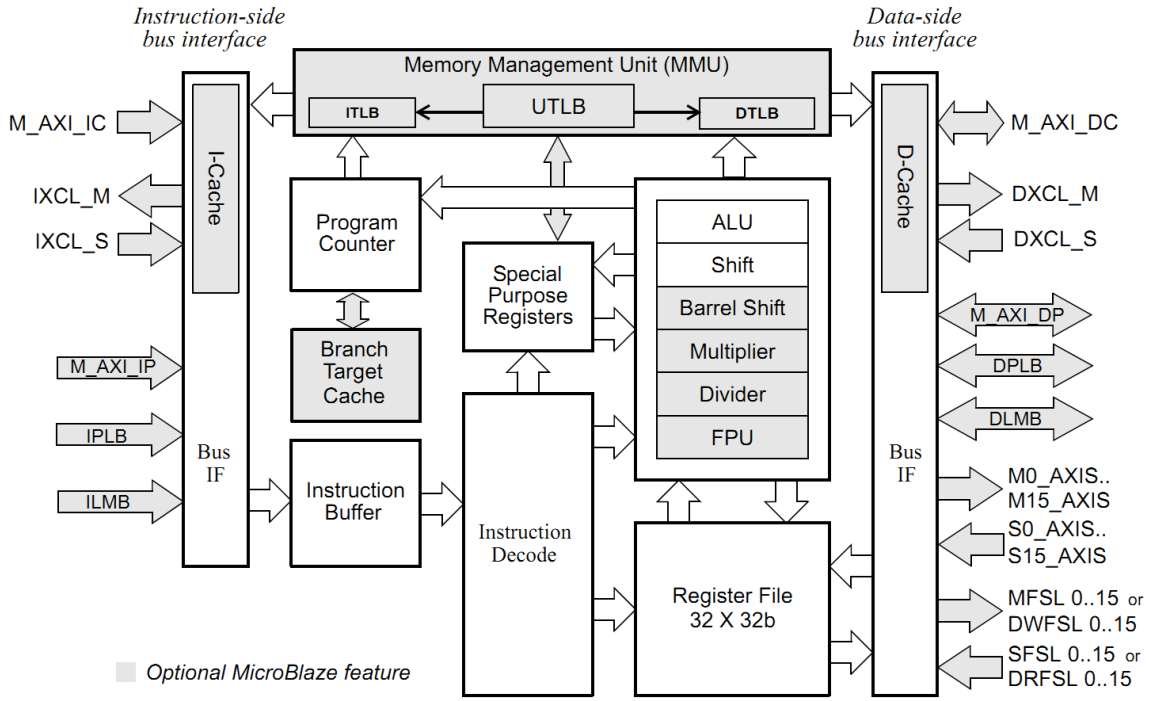


Figure 2.26: MicroBlaze Block Diagram [24]

Chapter 3

Related Work

As H.264 has become the standard for video encoding, efforts have moved towards optimizing the encoding process. Work done by Notebaert and De Cock [12] in 2004 demonstrates that portions of the encoding process can be performed faster in hardware than software due to parallelization which can complete many operations in one clock cycle. At the same time, the H.264 encoding process also contains portions which are sequentially oriented lending themselves to be performed more efficiently in software than hardware. In [12], the authors used an FPGA with a separate processor connected via a CardBus interface. Through their work, they identified communication between the CPU and the FPGA as the most important factor in hardware acceleration of the encoding process. By using an FPGA development board with an on-board CPU, the communication time between the CPU and hardware accelerators should be reduced in comparison to operating via a CardBus interface.

In 2008, Moorthy and Ye [9] targeted motion estimation as their primary subject for hardware acceleration. They designed a scalable motion estimation architecture using a Xilinx Virtex 5 FPGA. Their approach utilized Pixel Processing Units (PPUs) and the Propagate Partial SAD architecture. Each PPU was responsible for computing the costs of the 41 different potential motion vectors for the current pixel. After these costs were determined, the costs were compared using the Propagate Partial SAD methodology to determine which motion vector had the lowest cost. The design scaled by adding additional PPUs to process higher resolution frames. Figure 3.1 shows an example block diagram of the architecture making use of four PPUs. By using a single PPU at a 200 MHz clock, the implementation was

able to process VGA (640x480) frames at 28 FPS. This design was scaled up to 16 PPUs to process 1080i (1080x1088) frames at 62 FPS.

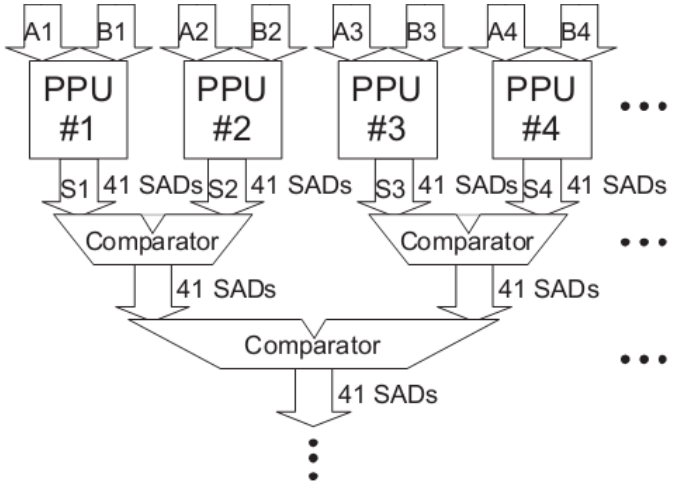


Figure 3.1: Moorthy et al. Motion Estimation Block Diagram [9]

Similarly, in 2010, Kao, Wu, and Lin [6] focused their efforts on fractional motion estimation. Their observations showed that fractional motion estimation consumed 40% of the H.264 encoding process. The authors designed a parallelized three engine accelerator to handle the various block sizes required for motion estimation. The first engine was responsible for 4x4 and 8x8 blocks while the second engine processed 8x4 and 4x8 blocks and the last engine handled the remaining block sizes of 8x16, 16x8 and 16x16. These three engines operated in parallel to efficiently determine the fractional motion vector and associated minimum cost for each pixel. A block diagram of their design is shown in Figure 3.2. The design was capable of processing one macroblock in an average of 631 cycles using a 154 MHz clock. At this rate, the design was able to process 1080i (1080x1088) video at 30 FPS.

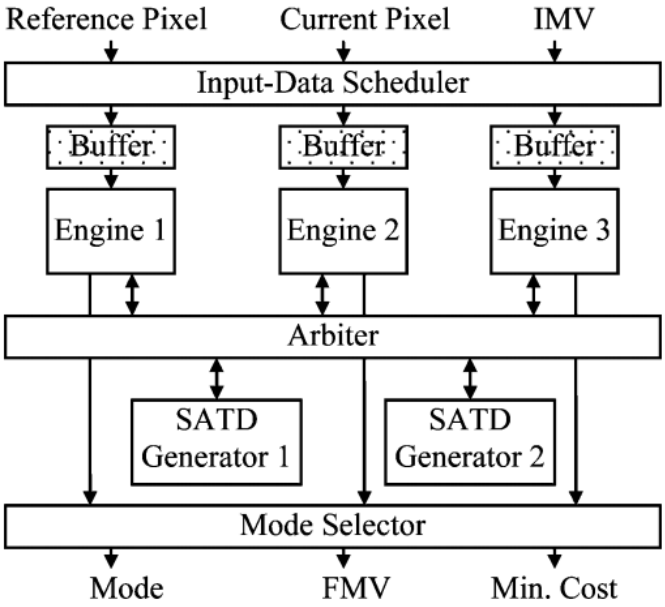


Figure 3.2: Kao et al. Fractional Motion Estimation Block Diagram [6]

In 2011, Ndili and Ogunfunmi [11] introduced their HMDS motion estimation architecture. The block diagram for this architecture is shown in Figure 3.3. The design uses an Address Generation Unit (AGU) to fill the current macroblock memory unit and two memory banks with candidate macroblocks. Two memory banks are used to efficiently read each 128 bit macroblock over a 64 bit bus. Processing Units (PUs) are utilized to compute the SAE costs for each of the 16 4x4 blocks of the candidate macroblock. Next, the SAD combination tree computes the cost of all 41 possible macroblock and sub macroblock candidates given the baseline 16 4x4 blocks. Lastly, the Comparison Unit, composed of 41 Comparison Elements (CEs) compares the cost of each mode across the current four candidates with the previously determined best candidate to determine the candidate with the optimal cost. This candidate and corresponding cost is then stored in a register and sent to external memory. The architecture is looped 10 times to examine a total of 40 candidate macroblocks. The authors observed a maximum clock speed of 246.5 MHz capable of computing CIF (352x288) frames in real time at 30 FPS using a search range of 16 with five reference frames.

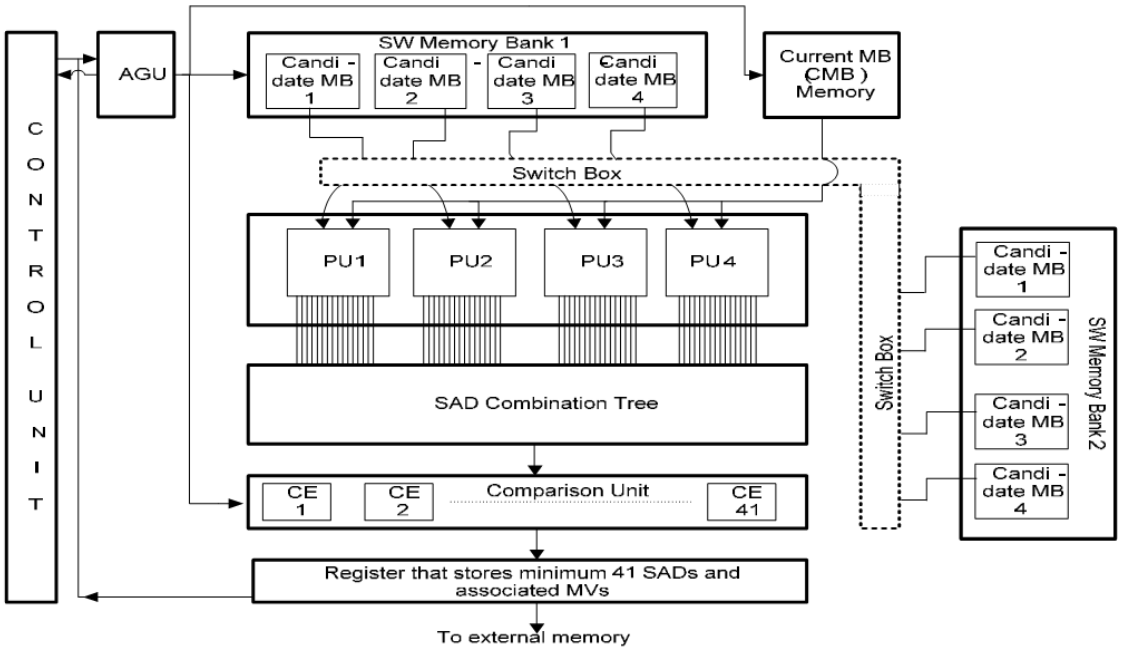


Figure 3.3: Ndili and Ogunfunmi Proposed HDMS Architecture Block Diagram [11]

In addition to motion estimation, other portions of the encoding process such as transforms, lend themselves to be implemented in hardware. In 2009, Owaida et al. [14] identified the 4x4 DCT, 4x4 and 2x2 Hadamard transforms as modules which could be performed more efficiently in hardware. Each of these transforms was implemented using only add and shift operations. Through parallelization and the implementation efficiency of simple add and shift operations, the authors were able to able to encode Quad Full HD (3840x2160) resolutions at 150 FPS.

Diniz et al. [3] investigated the creation of an intra frame prediction hardware architecture in 2009 using a Xilinx Virtex-II Pro FPGA. Their work focused on providing throughput capable of real time encoding for 1080p video at 30 FPS while consuming minimal hardware area. A block diagram of their resulting architecture is shown in Figure 3.4 where “T/Q” indicates the Transform/Quantization phase and “IT/IQ” indicates the Inverse Transform/Inverse Quantization phase required for reconstructing neighboring macroblocks. As shown in the diagram, their architecture consists of four main components. The “intra neighbor” component provides neighboring

encoded macroblocks. Next, the "sample predictor" uses chrominance and luminance prediction methods to determine candidates for the current macroblock. Third, the "SAD mode decision" block calculates and compares the SAD/SAE costs for each candidate to determine which mode is optimal for encoding. Lastly, the "prediction memory" block stores samples in a cache for repeated use. The resulting pipelined architecture was found to be capable of processing 1080p (1920x1080) intra prediction frames in real time at 110 FPS.

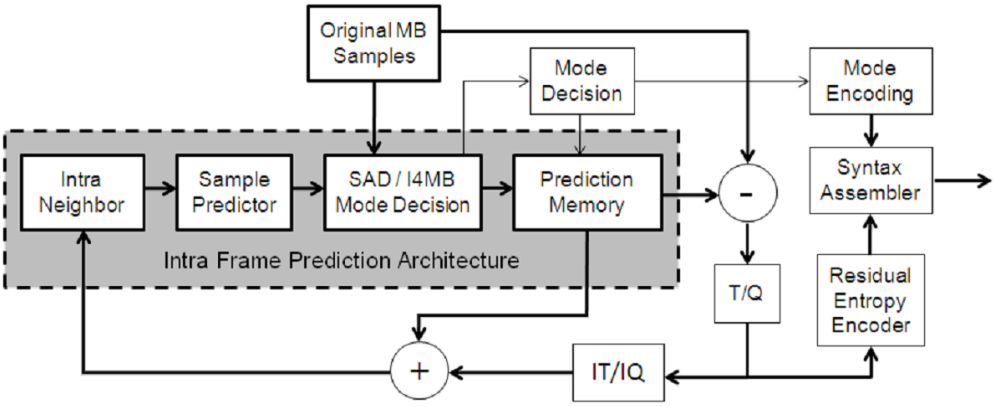


Figure 3.4: Diniz et al. Intra Prediction Block Diagram [3]

Efforts to decrease the bandwidth and power consumption required to transmit an H.264 encoded video have been examined by companies such as Ocean Logic [7]. In 2011, Ocean Logic determined that one of the best ways to reduce encoded video bandwidth requirements is to heavily compress reference frames. They developed a method to compress frames at rates ranging from 8-16:1. By decreasing the amount of storage required for each frame, the necessity for DRAM can be limited if not removed altogether allowing for low power and limited bandwidth designs.

This thesis builds specifically upon previous work [17] completed in 2011 for which the JM Reference Baseline encoder was stripped down to minimize executable size and increase readability of the code. This work was targeted at video encoding in bandwidth constrained, noisy environments. In [17], RoI encoding was included aimed at using variable bit rates with higher bit rates surrounding areas of interest and providing increased error resiliency.

Chapter 4

Design and Implementation

4.1 Software Baseline

To provide a reference point for comparison between the software only encoder and the designed hardware/software encoder, the JM Reference 17.2 H.264 Baseline encoder from [17] was modified to run in software on the Xilinx ML-507 development board. To provide maximum flexibility for future modifications and testing, the software baseline implementation was tested on both the PowerPC and MicroBlaze processors. The modifications to the encoder included primarily changes to timing operations, file Input/Output (I/O) operations, and the location of the input frame buffer.

Two hardware platforms, one for the MicroBlaze processor and one for the PowerPC processor, were configured using the Xilinx Embedded Development Kit (EDK) v13.1. Specifically, the hardware was configured using the Xilinx Platform Studio (XPS) tool while the software was configured with the Xilinx Software Development Kit (SDK).

4.1.1 Timing

Standard library timing functions were developed as Xilinx does not provide these functions for either of the two processors available on the XC5VFX70T FPGA. To create accurate timing functions, hardware timing registers and/or interrupts must be used. The PPC has a timing register built-in along with functions that allow for software access. The MicroBlaze processor does not include a timing register by default. To determine timing using the MicroBlaze processor, an XPS Timer was synthesized. Implementations for the standard C time function were developed for both the PPC and MicroBlaze

processors. The “time” function is used within the JM Reference encoder to assess encoding statistics. These statistics were used to assess the changes associated with adding hardware accelerators to the encoder.

4.1.2 File I/O

One of the major differences between running the encoder on a standard x86 desktop versus on the embedded processors of XC5VFX70T is the lack of a standard file system. The JM Reference encoder uses a mixture of standard library and Portable Operating System Interface for Unix (POSIX) methods for file I/O. By default, Xilinx includes non-functional stub functions in place of the standard library file I/O functions. To provide simple and flexible file access, the XC5VFX70T was interfaced with an x86 desktop via serial connection. An API was developed to open, read, write, seek, and close binary and text files from the embedded system. To accomplish this, an application written in C# was written to run on a desktop computer and listen for requests over the serial interface. In addition to listening for file I/O requests, the program was written to display standard output from the embedded system (which also utilizes the serial interface). A screenshot of the designed C# application is shown in Figure 4.1.

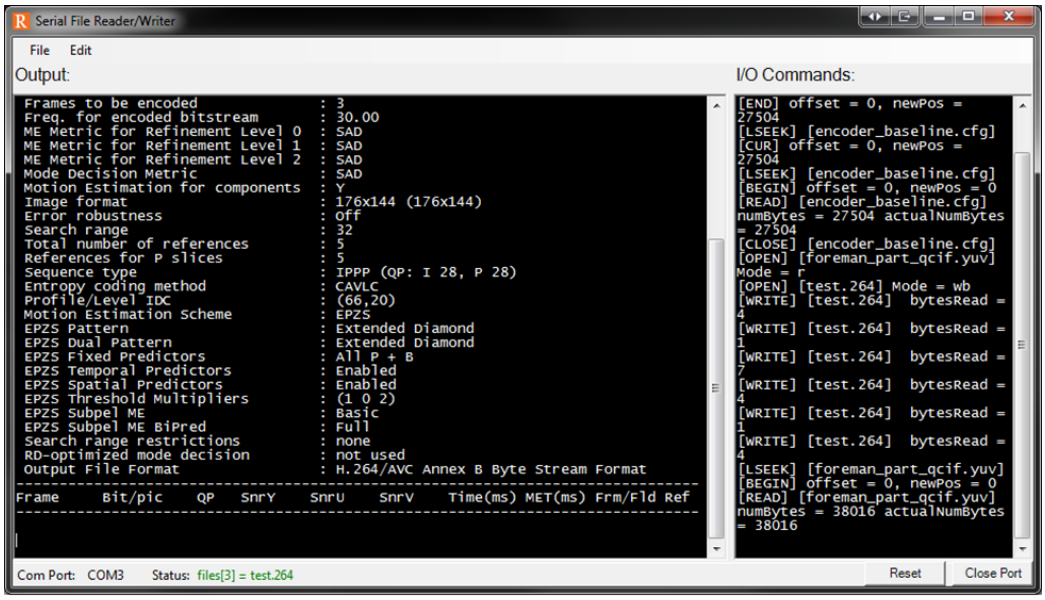


Figure 4.1: Serial File Reader/Writer C# Application Screenshot

Due to the speed limitations of the serial link, the option to use a compact flash card for reading the source file and writing the encoded file was added. The compact flash card is queried first for opening source files. If the file does not exist on the card, the file is requested over serial. The "WriteEncodedFileToCF" parameter was added to the encoder configuration file such that when enabled encoded files are written to the compact flash card, otherwise, they are written out over serial. Xilinx includes limited methods to access the compact flash card including open, read, write and close. These methods are sufficient for the source and encoded files as they are read/written sequentially from beginning to end without need of moving backwards within the file or more complex operations.

All standardized file I/O functions in the encoder were modified from the standard function names to use the matching Application Programming Interface (API) methods. A list of the original standard C / POSIX method names and their corresponding API methods are shown in Table 4.1. To account for the increased latency of pushing file I/O operations over a serial cable, timer calls were placed at the beginning and end of each API function to accumulate the total time spent on file I/O. At the end of the encoding process, the file I/O time can be subtracted from the total encoding time to provide consistent timing data independent of the speed of the serial link.

4.1.3 Input Frame Buffer Location

To further emulate a real time embedded system, an abstraction layer was added such that frames are read from a fixed location in memory rather than directly loaded from a file. This memory location is populated using frames from an input file via the aforementioned file I/O API. However, the idea is that in a real time system, the memory location will be populated by a device such as a camera. A diagram demonstrating the flow of frame data into the encoder is shown in Figure 4.2.

Original Method Name	Method Name	Description
open	sOpen	POSIX method to open a file
fopen	sFopen	Standard C method to open a file
fclose	sFclose	Standard C method to close a file
write	sWrite	POSIX method to write to a file
fwrite	sFwrite	Standard C method to write to a file
read	sRead	POSIX method to read bytes from a file
fread	sFread	Standard C method to read bytes from a file
lseek	sLseek	POSIX method for seeking to an offset within a file
fseek	sFseek	Standard C method for seeking to an offset within a file
fgets	sFgets	Standard C method for reading characters from a file
fputc	sFputc	Standard C method for writing characters to a file
fscanf	sFscanf	Standard C method for reading a file with a specified format
fprintf	sFprintf	Standard C method for writing to a file in a specified format

Table 4.1: File I/O API Method List

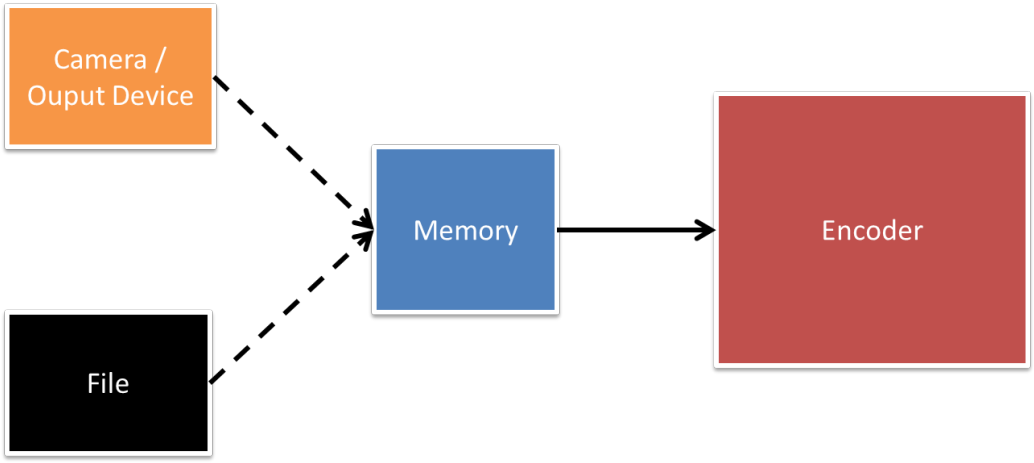


Figure 4.2: Dataflow of Input Frame from Source to Encoder

4.2 Software Analysis

The bottlenecks of the software encoder were analyzed using statistics generated from the reference encoder in addition to the timing statistics created using the gprof tool included in the Xilinx design suite.

4.2.1 Reference Encoder Statistics

The total time required for the encoding process as well as the total time required for motion estimation is tracked by the JM Reference encoder. Motion estimation is widely recognized as the most computationally expensive segment of video encoding. To assess the cost within the JM Reference encoder, timing statistics for each of the five motion estimation schemes included in the JM Reference encoder were gathered on the PPC using 100 frames of the Foreman video sequence with CIF resolution encoded with one reference frame. The results are shown in Table 4.2.

Motion Estimation Method	ME Time (s)	Total Time (s)	Percent ME Time (%)
Full	3519.967	3733.290	94.286
Fast Full	2494.316	2697.696	92.461
Hex	59.230	279.546	21.188
Simple Hex	39.433	256.768	15.357
EPZS	32.025	247.707	12.929

Table 4.2: Motion Estimation Timing Data (PPC with 400 MHz Clock)

The results demonstrate that a minimum of about 15% of the total encoding process is spent on motion estimation with upwards of 90% of the total encoding time being spent with the full motion estimation schemes. Note that these results are for motion estimation using one reference frame. The total motion estimation time scales approximately proportionate to the number of references frames used. This is due to the fact that the motion estimation algorithm is looped over each available reference frame. For example, if 5 reference frames were used with the EPZS algorithm, the percentage of motion estimation time increases to roughly 40%.

4.2.2 Gprof Statistics

The gprof tool was used to collect statistics regarding the number of times a function is called and the duration of each call. Gprof was run using the first three frames of the Foreman video sequence encoded with EPZS motion estimation and one reference frame. The resulting table produced by gprof contains statistics on about 5,000 functions. To narrow the table down to a reasonable number of functions for further analysis, the table was sorted in descending order based on the time the function consumes per call. A lower number of calls minimizes context switching between hardware and software and reduces the associated communication overhead. By sorting on this criteria, the top functions have relatively few calls while contributing a significant portion of the overall encoding time. The resulting top 15 functions are shown in Table 4.3.

#	Function Name	Calls	Time/Call (ms)	%Time
1.	getHorSubImageSixTap	3	14.5	0.521
2.	getVerSubImageSixTapTmp	3	12.7	0.456
3.	getVerSubImageSixTap	3	12.5	0.450
4.	getVerSubImageBiLinear	9	6.39	0.691
5.	getDiagSubImageBiLinear	3	6.22	0.224
6.	getHorSubImageBiLinear	9	5.77	0.624
7.	getSubImageBiLinear	15	5.40	0.974
8.	ParseContent	1	2.23	0.03
9.	Init_Motion_Search_Module	1	1.90	0.013
10.	compute_SSE	18	1.68	0.04
11.	generateChromaXX	294	1.62	5.709
12.	generateChroma10	42	1.55	0.532
13.	generateChroma01	42	1.11	0.561
14.	EPZSSliceInit	2	0.87	0.021
15.	getSubImageInteger_s	3	0.64	0.023

Table 4.3: Gprof Statistics (PPC with 400 MHz Clock)

Functions 1-7 and 15 comprise the major functions of luminance prediction. Functions 8 and 9 are part of the encoder initialization phase and do not apply to the actual encoding process. Function 10 is an error evaluation function used for evaluating encoder quality statistics. Functions 11-13 represent the diagonal, vertical, and horizontal chrominance prediction

methods. Lastly, function 14 is specific to the EPZS motion estimation algorithm. Note that motion estimation methods do not rise to the top of this table because the motion estimation algorithms use many different small methods which alone do not constitute a significant amount of the total encoding time.

4.2.3 Software Analysis Conclusions

The encoder statistics and gprof results indicate that the intra prediction and motion estimation stages of the encoder compose the highest percentage of encoding time. Due to the smaller frame size of chrominance and the simplicity of chrominance prediction in comparison to luminance prediction, chrominance prediction was selected to be accelerated in hardware. Dependent on the number of reference frames used, motion estimation can dominate the encoding process. To help alleviate this computational load on the CPU, motion estimation was also chosen to be accelerated in hardware.

4.3 Hardware Accelerators

4.3.1 Hardware Accelerator Interface

As discussed in [12], efficient communication between the CPU and hardware becomes essential to reap the full benefits of a hardware/software system. To minimize communication time when switching between hardware and software and vice-versa, dual-port BRAM was chosen to be the primary interface. Hardware accelerators connect directly to one port of BRAM while the CPU communicates with the second port over the PLB. In addition to using BRAM as the primary accelerator interface, software accessible registers were selected to enable and disable as well as communicate the current state of the accelerator. These software accessible registers can be read and written to directly from hardware or over the PLB from software. Figure 4.3 depicts this hardware/software interface.

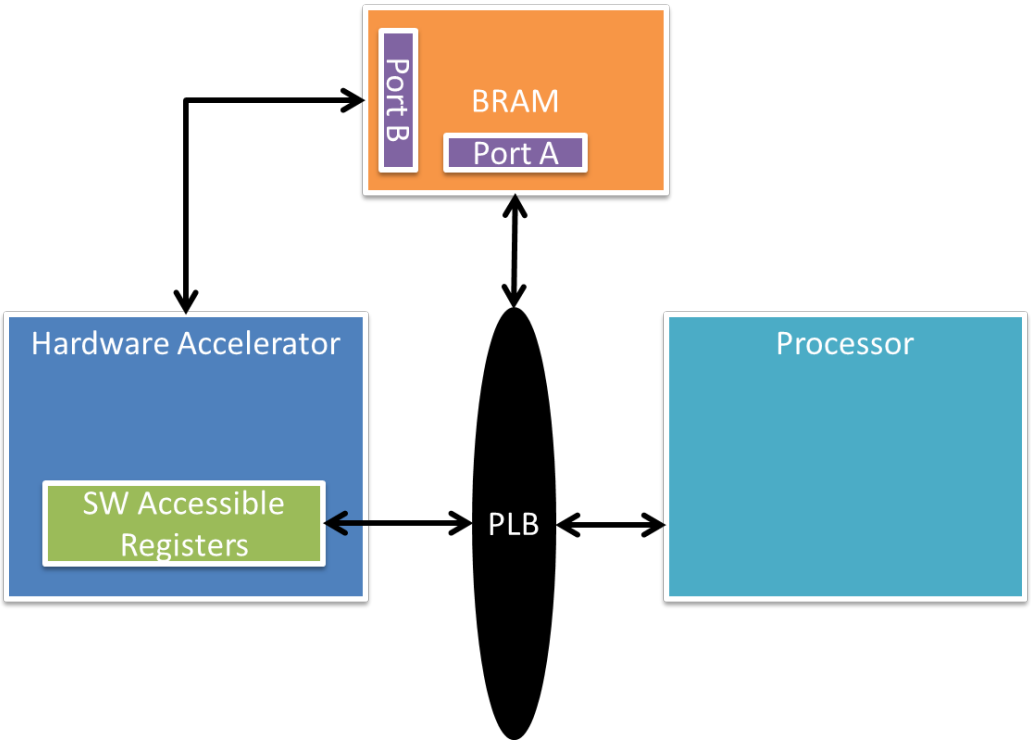


Figure 4.3: Hardware/Software Interface

4.3.2 Chrominance Prediction Accelerator

The chrominance prediction hardware accelerator was created such that it is functionally equivalent to the respective software methods. The JM Reference encoder performs chrominance prediction slightly different than described in the H.264 standard while the result is the same. Rather than operate on individual macroblocks, the Reference encoder operates on the entire frame. Furthermore, the Reference encoder pads all sides of the frame to facilitate logic for later stages of the encoder. Figure 4.4 breaks the resulting frame into nine sections with E as the actual frame while the remaining eight blocks are padded sections. By default, the encoder pads nine pixels vertically and sixteen pixels horizontally forcing blocks A, C, G, and I to be 16x9.

Operating on whole frames requires a significant amount of BRAM in comparison to operating on individual macroblocks and limits the maximum resolution to that which can fit in BRAM. However, whole frame operation prevents redundant computation and allows for large memcpy operations

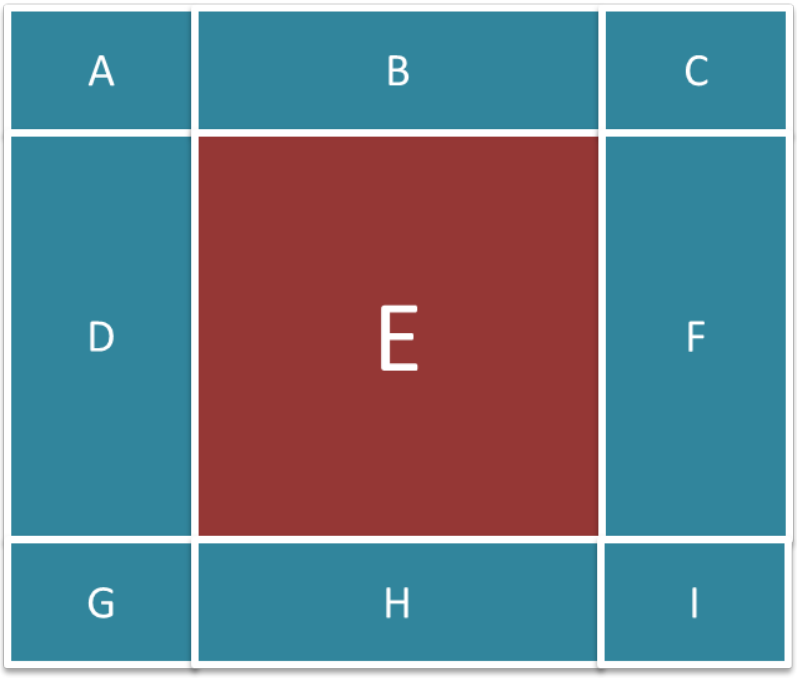


Figure 4.4: Chroma Frame Padding

which can decrease the overall execution time. The Reference encoder computes 64 different predicted chrominance frames per each of the two source chrominance frames. The 64 prediction frames are computed from the combination of 49 plane predicted frames, 7 horizontal predicted frames, 7 vertical predicated frames and 1 DC predicted frame. Note that multiple frames can be computed using the same method by varying the weights. Figure 4.5 demonstrates the flow of data into and out of the chrominance prediction method in the JM Reference encoder.

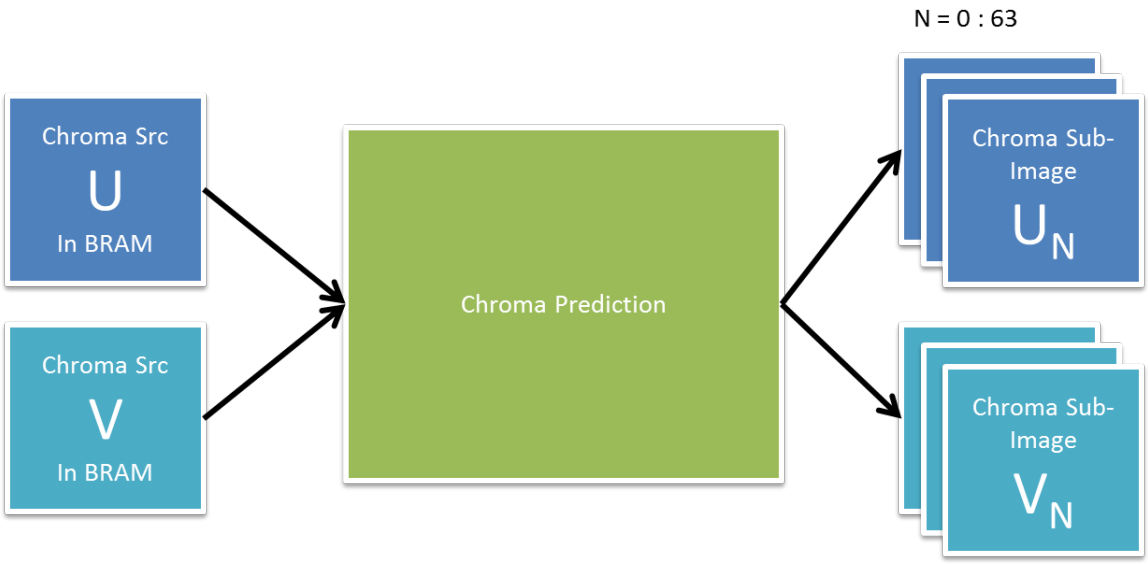


Figure 4.5: Chrominance Prediction Data Flow Diagram

The designed hardware accelerator performs horizontal, vertical, and plane chrominance prediction. The decision to forgo DC prediction was made as the primary software operation is memcpy, rather than mathematical computation. The accelerator operates on source data from left to right from the top to the bottom of the frame. This strategy of reading frame data provides for sequential reads of memory without having to jump forward to the next row as would be required in top to bottom operation and thus decreases the complexity of determining the next memory location.

Regardless of the pattern used to read source data, writing new data proves problematic as the width of source data available is the same as the width of data which can be written. When performing horizontal and plane prediction, the previous and current source pixels are used to determine the resulting destination pixel. Thus, if the number of available source pixels is N, then the number of destination pixels which may be written at one time is $N - 1$. To overcome this issue, memory buffers were added. These memory buffers hold a total of $2N$ source pixels and require one extra cycle to fill. When a new memory address is accessed, the memory buffer purges memory from two addresses ago and maintains the data from the current and previous addresses. Note that the extra cycle required to fill the

buffer is only necessary when accessing a non-sequential address. Thus, the memory buffer only requires an extra cycle at the start of each chrominance frame as all source frames are accessed sequentially. By adding a memory buffer twice the width of memory, the full N memory width can be written each cycle.

In addition to the byte accessible memory obstacle, an efficient chrominance prediction accelerator requires simultaneous access to pixels from the current row and the following row to determine pixels for vertical and plane prediction. To overcome this issue, a second bank of BRAM which mirrors the first bank of source memory was added.

Ideally, the frames generated from the chrominance accelerator would be left in BRAM until the next source frame is read. However, for data to be retained in BRAM, a total of 128 padded frames would need be stored in BRAM (64 padded frames for each of the two chrominance components). Assuming these padded frames are generated from QCIF source frames, 1,367,040 bytes (see eq. (4.1)) of BRAM would be required. The XC5VFX70T contains 144 36Kbit BRAMs capable of holding 648,000 bytes at one time. Therefore, even if all BRAM on the XC5VFX70T was exhausted, QCIF chrominance predicted frames cannot not remain in BRAM and thus must be copied from BRAM to main memory which in this case is DDR2. This constraint heavily influenced further design decisions. Rather than synthesizing more hardware to compute chrominance prediction frames in parallel, the decision to compute one chrominance predicted frame at a time was made. This is due to the bottleneck of copying resulting frames from BRAM to DDR2 over the PLB. To help alleviate the additional demands of this constraint, a second BRAM destination bank was added such that the accelerator can begin computing new frame data while the previous frame data is copied over the PLB. This addition essentially eliminates the cost of computing chrominance frames (except for generating the first frame) as the cost of hardware computation is substantially less than the cost of copying frames over the PLB.

$$
\begin{aligned}
TotalBRAM = \; & NumFrames * (\frac{SrcWidth}{2} + 2 * PadWidth) * \\
& (\frac{SrcHeight}{2} + 2 * PadHeight) \\
TotalBRAM = \; & 128 * (\frac{176}{2} + 2 * 16) * (\frac{144}{2} + 2 * 9) \\
TotalBRAM = \; & 1,367,040 \text{ Bytes} \qquad (4.1)
\end{aligned}
$$

The designed accelerator uses a state machine which breaks down each of the three chrominance prediction methods into nine sections as presented in Figure 4.4. The cycle count equations for the horizontal, vertical and plane prediction methods as well as the total cycle count are shown in eqs. (4.2) to (4.5) respectively where PH indicates padding height, PW indicates padding width, H indicates source height, and W indicates source width. Thus, the total number of cycles required to compute all 63 chrominance predicted frames given one QCIF source chrominance frame (88x72) with a padding width of 16 and padding height of 9 according to eq. (4.5) is 281,148 cycles. After synthesis results were obtained, it was found that the maximum supported clock of the accelerator is 100 MHz. Thus, the accelerator could process a maximum of 355 QCIF chrominance FPS.

$$\begin{aligned}
\text{HCycleCount} = \; & (2PH+1)(\frac{PW}{4}+6+3\frac{W}{8})+ \\
& (H-1)(\frac{PW}{4}+4+3\frac{SW}{8})+1 \qquad (4.2)\\
\text{VCycleCount} = \; & (2PH+1)(\frac{PW}{4}+6+3\frac{W}{8})+ \\
& (H-1)(\frac{PW}{4}+4+3\frac{SW}{8})+1 \qquad (4.3)\\
\text{PCycleCount} = \; & (2PH+1)(\frac{PW}{4}+6+3\frac{W}{8})+ \\
& (H-1)(\frac{PW}{4}+4+3\frac{SW}{8})+1 \qquad (4.4)\\
\text{TotalCycleCount} = \; & 7(\text{HCycleCount})+7(\text{VCycleCount})+ \\
& 49(\text{PCycleCount}) \qquad (4.5)
\end{aligned}$$

A block diagram of the final chrominance prediction accelerator is shown in Figure 4.6. Note that En, Reset, SrcAddr, DstAddr, PadWidth, PadHeight, SrcWidth, SrcHeight, PassNum, MemBankRead, and Complete represent signals accessible through software registers.

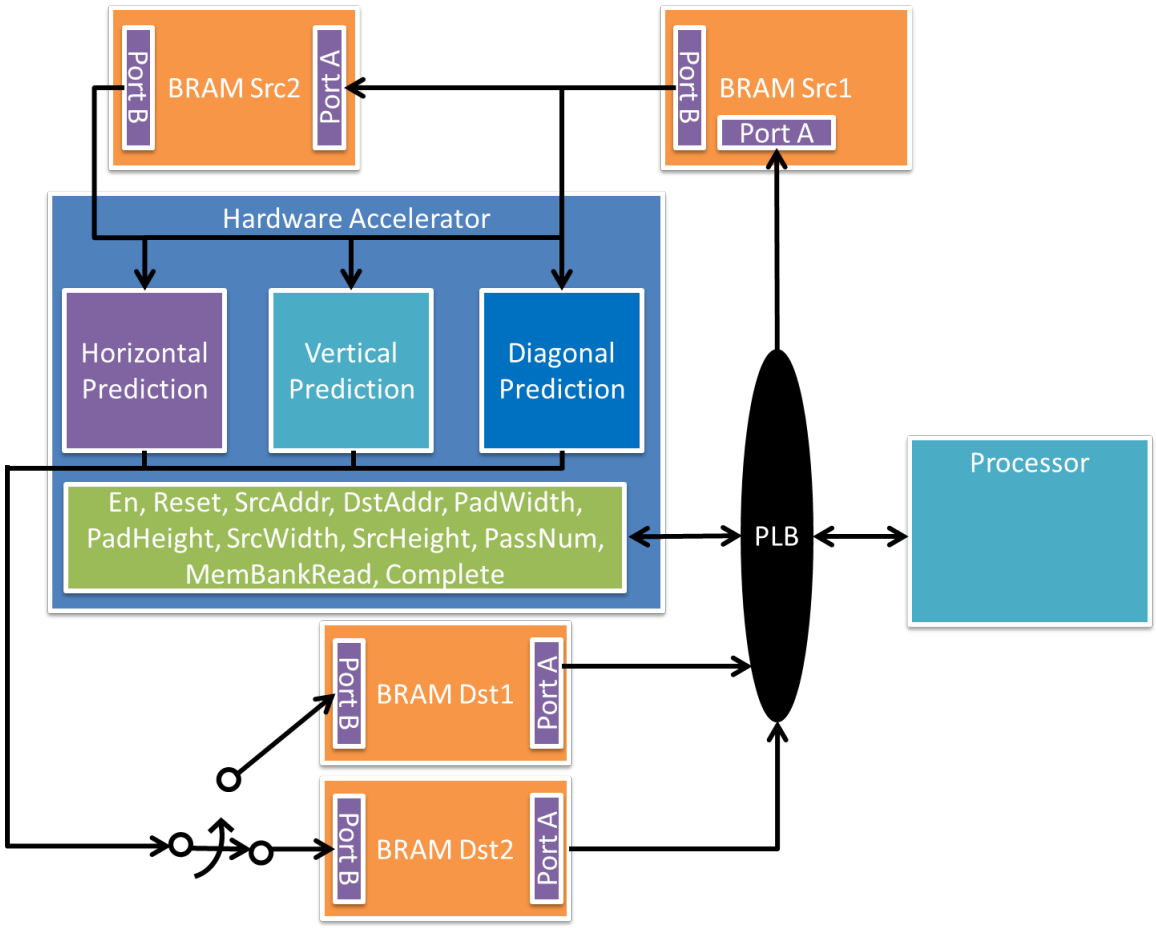


Figure 4.6: Chrominance Prediction Hardware Accelerator Block Diagram

Pseudocode for operating the accelerator from software is shown in Table 4.4

```
for Each Chrominance Frame do
    Reset Accelerator
    Configure Accelerator Frame Parameters
    Copy Source Chrominance Frame to BRAM Src1
    Enable Accelerator
    while Accelerator Is Not Complete do
        Copy Generated Frames From BRAM Dst to Main Memory As They Are Completed
    end while
end for
```

Table 4.4: Pseudocode for Software Communication with Chroma Prediction Accelerator

4.3.3 Motion Estimation Accelerator

The designed motion estimation accelerator is based off of the HMDS architecture proposed in [11]. Some notable changes were made to the architecture in order to adapt to the available hardware, increase resulting quality and allow greater flexibility.

In [11], the authors use two 64-bit wide software memory banks, each containing four candidate macroblocks. The XC5VFX70T's 128-bit wide BRAM banks were substituted to allow reading of one row of a macroblock at a time, thus reducing the number of cycles required to load processing units by half. By providing two separate memory banks, one memory bank can be populated while the other is processed. To extend this idea, using dual port BRAM, one port can be used to populate a separate address space while the current address space is processed. Since the BRAM used has a 128-bit width, the minimum size BRAM block on the XC5VFX70T is 16KB, allowing for a minimum of 64 macroblocks to be held in memory at one time.

The proposed architecture specifies that motion cost should be determined through SAE calculations for each candidate macroblock and then sent to the comparison unit to determine the optimal motion vector. The resulting optimal motion vectors and their respective SAE values for each block type are then sent to the CPU where the cost of encoding each motion vector is added to the SAE cost to determine the optimal block type. This method is sufficient for finding a suitable motion vector, but yields relatively high bit rates in comparison to software motion estimation methods. Mainstream software methods add the extra bit cost of encoding motion vectors to each SAE value before comparison rather than adding this cost solely to the optimal vectors. To move this additional cost estimate before the comparison unit in the HMDS architecture, an additional rate distortion unit was added. This unit requires the motion vectors for each candidate macroblock, the predicted motion vector of the current macroblock, and the encoding lambda value to predict the motion vector encoding cost. Candidate motion vectors are stored in dual port BRAM as they change with each pass, while lambda and the predicted motion vector is set through a

software accessible register. The addition of the rate distortion unit does not add any additional cycles to the design as the processing units and distortion unit can run at the same time. The architecture for the rate distortion unit is shown in Figure 4.7 where CandMVs represents the current four candidate motion vectors, PredMV represents the predicted motion vector for the current macroblock, and CandMVCosts represents the resulting predicted cost of encoding each candidate motion vector. The MV Bit LUT is used to compute the result of eq. (4.6) where x is the delta motion vector component value and $MVBits$ is the number of additional bits required to encode the component.

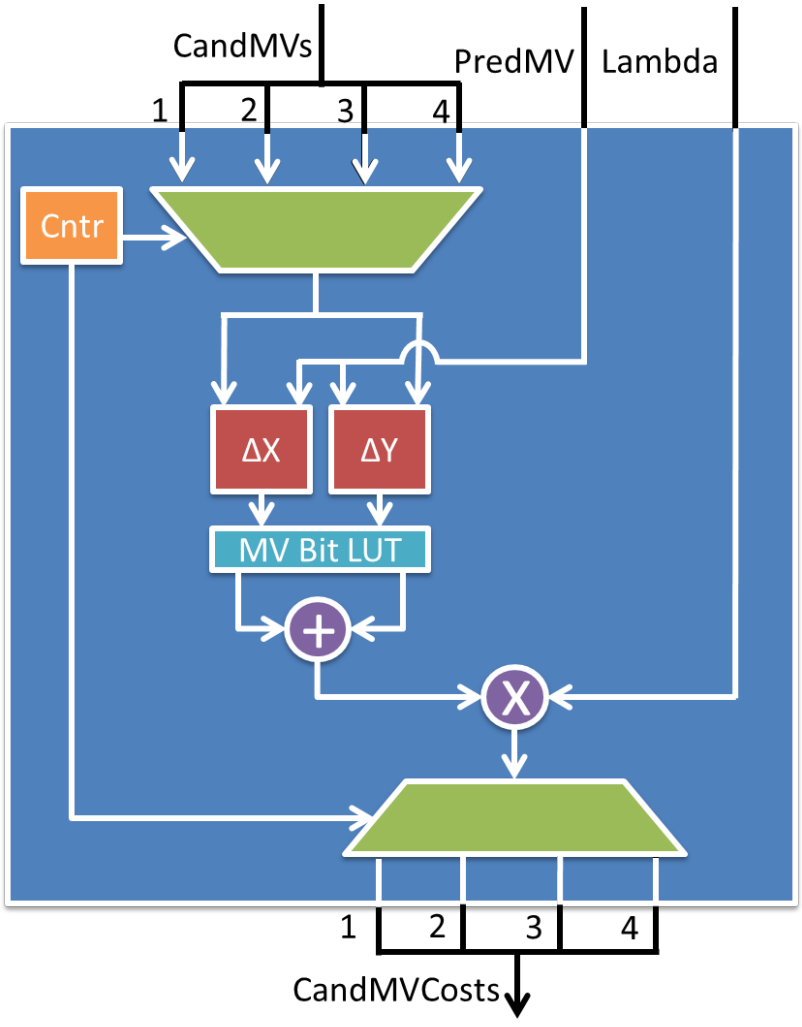


Figure 4.7: Rate Distortion Unit Architecture

$$MVBits = \begin{cases} 2 * log_2(x) + 3 & \text{when } x > 0 \\ 1 & \text{when x = 0} \end{cases} \tag{4.6}$$

The output of the rate distortion unit is sent to the comparison unit where it is added to the current SAE value prior to comparison. One extra cycle in the comparison unit is required for addition of the SAE cost with the encoded motion vector cost. The restructured comparison unit is shown in Figure 4.8.

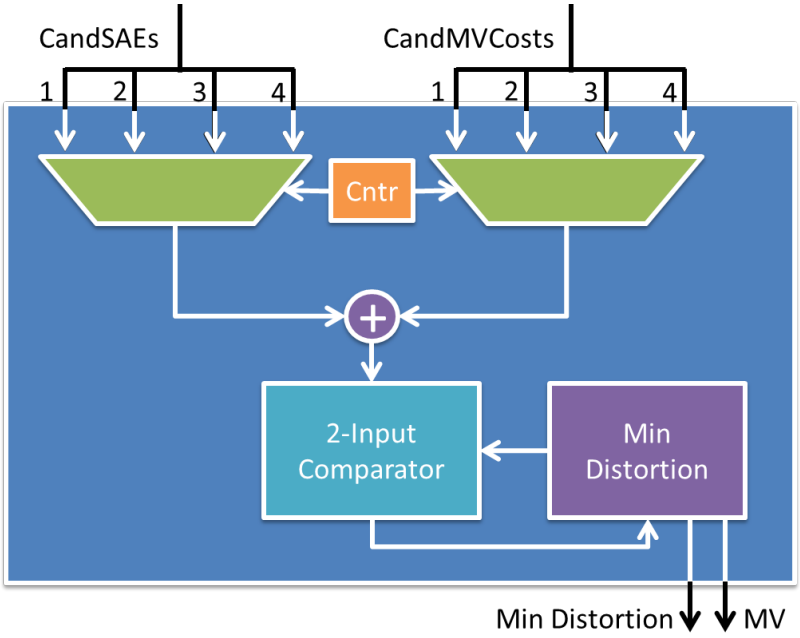


Figure 4.8: Comparison Unit Architecture

To further increase flexibility of the accelerator and account for the limited amount of BRAM available, the address generation unit was moved from hardware to the processor and the number of passes required for completion was made configurable. Relocation of the address unit allows the processor to move candidate macroblocks into BRAM from main memory. Furthermore, this allows flexibility as to which motion estimation algorithm is used. The architecture presented in [11] is primarily a method for determining and comparing distortion values for four candidate macroblocks at a

time. Moving the address generation unit to the processor makes the hardware architecture motion estimation algorithm independent. The hardware accelerator can then be used for algorithms such as Full, UMHex, Simplified UMHex and EPZS search. To prove the concept of this additional ability, software was written for Full search in addition to HMDS.

To decrease execution time and increase quality, the motion vector prediction phase was moved from software to hardware. The proposed architecture suggests that the costs of the two predicted motion vectors, the zero vector and median vector should be computed in software and then used as the center of the first HMDS search pattern. To decrease the execution time required for software to compute the motion costs of the two motion vector predictors, this computation was moved to the accelerator. Rather than waste two additional candidate blocks in the accelerator during this pass, two additional motion vector predictors were added and all four predictors are processed through the hardware accelerator to decrease execution time and increase quality. The four predictors evaluated are the zero vector, left neighboring vector, top-left neighboring vector, and the top neighboring vector.

To further decrease execution time, it was noted that the center of the last HMDS search pattern could be the center of the previous HMDS search pattern. In this case, the final passes would reevaluate the same locations. To prevent this reexamination, software instructions were added to check whether the center location of the last pattern matches that of the previous pattern and terminates the search if applicable.

The total number of cycles required for motion estimation using the accelerator can be computed using eq. (4.7). Thirty-one cycles are required to compare each set of four macroblocks. Once all macroblocks have been compared, ten cycles are required to write the results to memory. In the worst case scenario, the two HMDS diamonds have different centers and thus 44 macroblocks are examined. In the best case scenario, the two HMDS diamonds have the same center and thus 32 macroblocks are examined. A QCIF frame contains 99 macroblocks. Assuming the worst case scenario, a total of 4,356 macroblocks need to be evaluated versus 3,168 macroblocks in the best case scenario. Synthesis results yielded a maximum supported

clock of 227.8 MHz. If the max clock was used, the accelerator could process 52,295 QCIF FPS in the worst case scenario and 71,906 QCIF FPS in the best case. If 1080p (1920x1080) frames with a total of 8,100 macroblocks were processed, the accelerator could process 639 FPS in the worst case and 878 FPS in the best case. To facilitate hardware design, the motion estimation accelerator was set to match the PLB clock of 100 MHz. At 100 MHz, the accelerator can process 22,956 QCIF FPS in the worst case and 31,565 QCIF FPS in the best case.

$$\text{MECycleCount} = 31 * \frac{\text{\# of Macroblocks}}{4} + 10 \tag{4.7}$$

A block diagram of the final motion estimation accelerator is shown in Figure 4.9. En, Reset, MemBankReady, Lambda, CurMV, PassStopNum, PassNum, and Complete represent signals accessible through software registers.

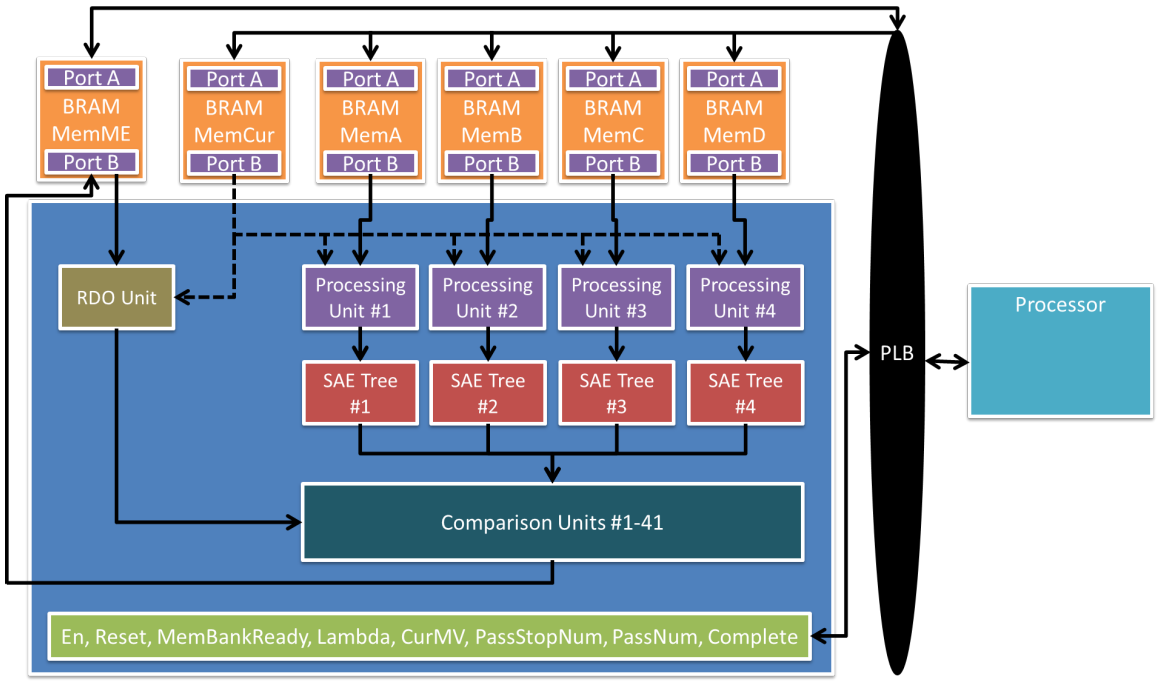


Figure 4.9: Motion Estimation Accelerator Architecture

Pseudocode for operating the accelerator from software using the HMDS motion estimation algorithm is shown in Table 4.5 while pseudocode for operating the accelerator for Full Search motion estimation is shown in Table 4.6.

```
function HA_HMDS
Reset Accelerator
Configure Accelerator Parameters (Set PassStopNum to 11)
Copy Current Macroblock to BRAM
for Each Macroblock in Motion Vector Predictor Phase do
   Copy Candidate MV Macroblock
end for
Enable Accelerator
Read center value from BRAM
for Each Macroblock in First HMDS Pattern located at center do
   Copy Candidate MV Macroblock
end for
Read new center value from BRAM
for Each Macroblock in First Large Diamond Pattern located at center do
   Copy Candidate MV Macroblock
end for
Read new center value from BRAM
if New center value != Previous center value then
   for Each Macroblock in Second Large Diamond Pattern located at center do
      Copy Candidate MV Macroblock
   end for
end if
while Accelerator not complete do

end while
Read optimal distortion and MV values from BRAM
```

Table 4.5: Pseudocode for HMDS Software Communication with Motion Estimation Accelerator

function HA_Full
Reset Accelerator
Configure Accelerator Parameters (Set PassStopNum to (SearchRange*2+1)^2-1)
Copy Current Macroblock to BRAM
Copy First Four Candidate MV Macroblocks
Enable Accelerator
for Each Remaining Macroblock in Search Range **do**
  Copy Candidate MV Macroblock
end for
while Accelerator not complete **do**

end while
Read optimal distortion and MV values from BRAM

Table 4.6: Pseudocode for Full Search Software Communication with Motion Estimation Accelerator

Chapter 5

Testing

5.1 Verification

To verify the functionality of each hardware accelerator component, separate software programs were designed. The chrominance program reused the prediction functions included in the JM Reference encoder and was modified to process generated input vectors and store the resulting frames to file. The HMDS program was written to accept input vectors, perform the functionality of HMDS specified in [10], and store the results to file. To verify the functionality of the accelerators with their respective test programs, software projects were created using the Xilinx SDK to load test vectors into the accelerators and to write the results out to file in a format matching that of the test program file format. The resulting files generated from the test programs were compared with the corresponding hardware accelerator test files to verify full functionality.

5.2 Testing Environment

The chrominance and motion estimation accelerators were synthesized in two different hardware environments. The first environment, hereafter referred to as the PowerPC configuration, is shown in Figure 5.1 and uses the PPC 440 with a clock rate of 400 MHz, 32KB instruction and data caches, and a PLB clock rate of 100 MHz. The second environment, hereafter referred to as the MicroBlaze configuration, uses the MicroBlaze processor with a clock rate of 100 MHz, 32KB instruction and data caches, and PLB clock rate of 100 MHz as shown in Figure 5.2.

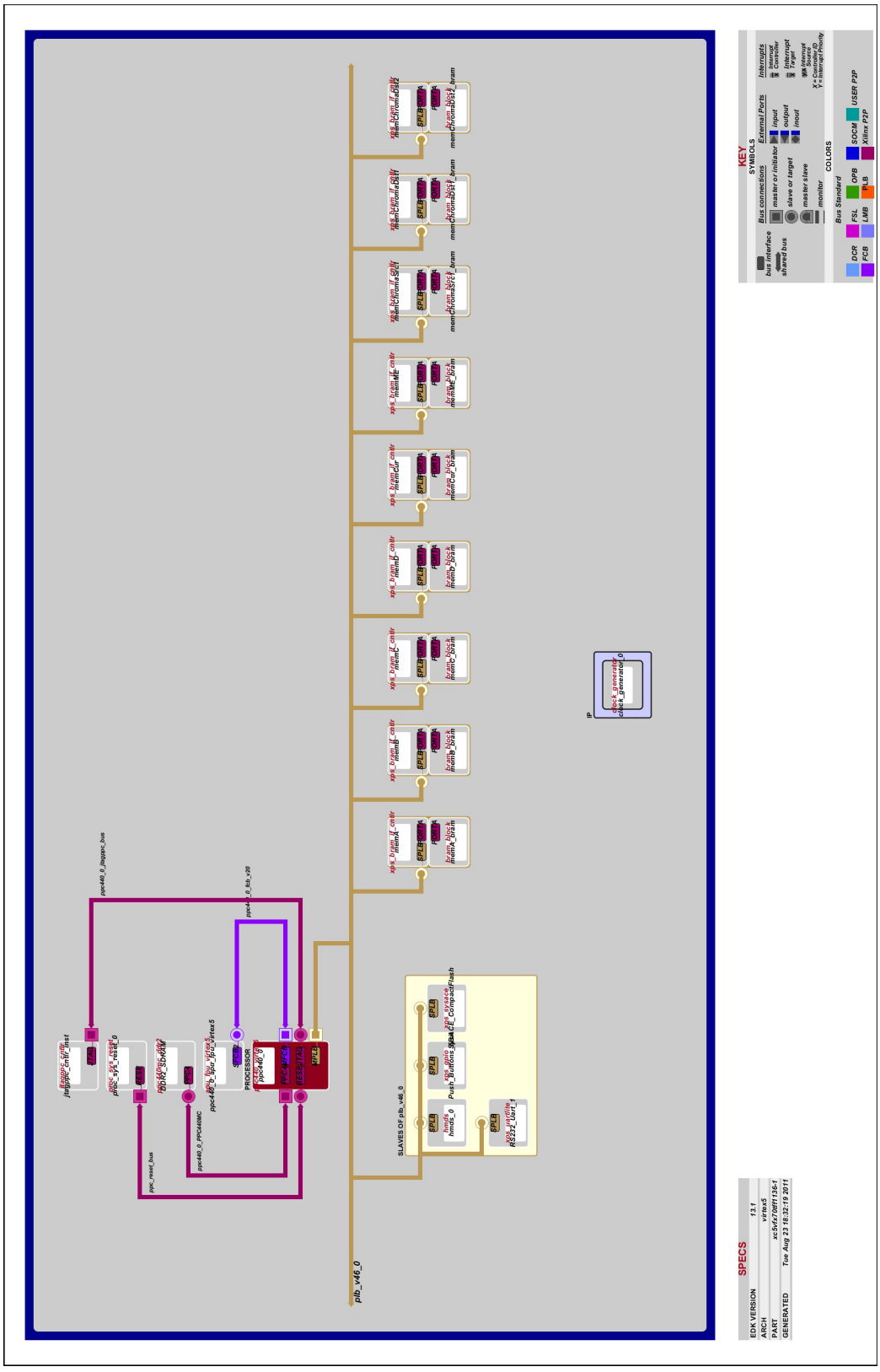


Figure 5.1: PowerPC 440 Test Environment

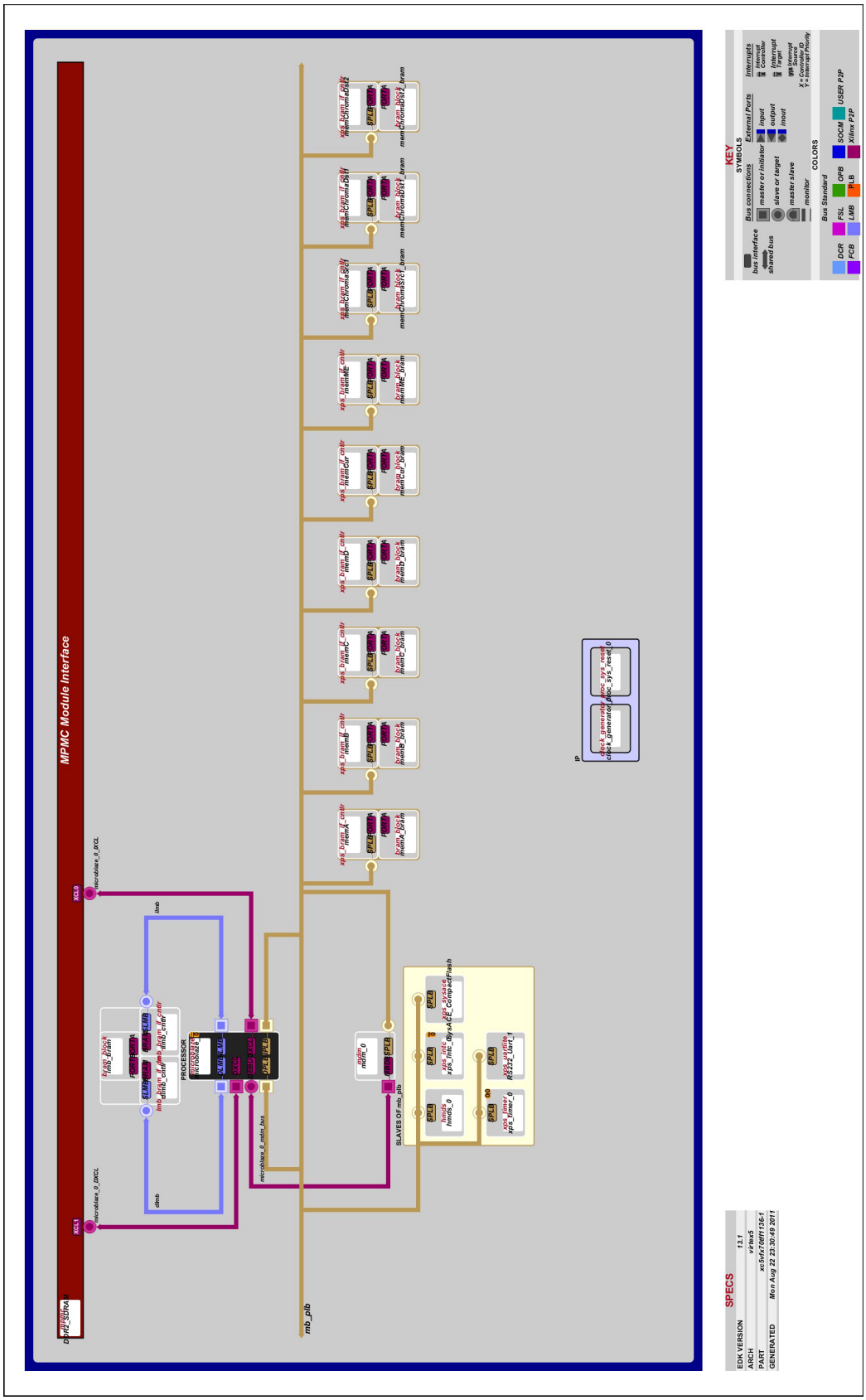


Figure 5.2: MicroBlaze Test Environment

5.3 Encoding Parameters

To facilitate comparisons between the designed motion estimation accelerator and the accelerator designed in [11], similar encoding parameters were used. All of the video sequences used were freely obtained from `http://media.xiph.org/video/derf/` and are shown in Table 5.1 along with the quantization parameters, search range, and number of frames used for testing. In addition, the following parameters remained constant throughout all tests unless otherwise specified: 30 FPS frame rate, CAVLC entropy encoding, full pel motion estimation, fast high complexity rd-optimized mode decision, IPPP... GOP Structure, 1 reference frame, and 15 second I-frame refresh rate.

Sequence Name	**QP**	**Search Range**	**Frame Size**	**Frame Count**
Foreman	22, 25, 28, 31, 33, 35	32	CIF	100
Mother-Daughter	22, 25, 28, 31, 33, 35	32	CIF	150
Stefan	22, 25, 28, 31, 33, 35	16	CIF	90
Flower	22, 25, 28, 31, 33, 35	16	CIF	150
Coastguard	18, 22, 25, 28, 31, 33	32	QCIF	220
Carphone	18, 22, 25, 28, 31, 33	32	QCIF	220
Silent	18, 22, 25, 28, 31, 33	16	QCIF	220
Suzie	18, 22, 25, 28, 31, 33	16	QCIF	150
Mobile	18, 22, 26, 30, 34, 38	16	SIF	220
Tennis	18, 22, 26, 30, 34, 38	16	SIF	150
Waterfall	18, 22, 26, 30, 34, 38	16	CIF	125
Container	18, 22, 26, 30, 34, 38	16	CIF	220
Hall Monitor	18, 22, 26, 30, 34, 38	32	CIF	220
Salesman	18, 22, 26, 30, 34, 38	16	QCIF	220
News	18, 22, 26, 30, 34, 38	16	QCIF	220
Miss America	18, 22, 26, 30, 34, 38	32	QCIF	150

Table 5.1: Test Video Sequences

Chapter 6

Results

6.1 Chrominance Accelerator

The designed chrominance accelerator consumes 1004 slice registers, and 1756 slice LUTs. The maximum supported clock frequency is 100 MHz. Due to limited availability of BRAM on the FPGA, SIF (352x240) is the maximum supported standardized frame size for chrominance acceleration.

Table 6.1 compares the total and average per frame chrominance prediction execution times with and without the chrominance accelerator enabled. Note that the number of computations for chrominance prediction is dependent only on the frame size; not the content of the frame. Thus, the average per frame results for the Miss America and Mobile sequences can be extended to all QCIF and SIF sequences respectively. The results show that it is more beneficial to use software than hardware for QCIF resolutions in the PowerPC configuration, yet for SIF resolutions it is slightly more beneficial to use hardware than software. This is likely due to the computation of padding in software. To pad a frame in software, only one row of padding is computed and the following padded rows are copied from the first row. This methodology is efficient and outweighs the benefits of copying the results over the PLB from the accelerator. However, the primary strength of the accelerator is computing frame pixels. Thus, as the frame size increases, the proportion of computed frame pixels to padded pixels also increases and the benefits of the accelerator become more apparent.

The PowerPC runs four times faster than both the PLB whereas the MicroBlaze runs at the same speed as the PLB. Therefore, the accelerator

yields greater speedups for the MicroBlaze than the PowerPC. By matching the PLB clock rate, the MicroBlaze is able to achieve speedups of 4.54 and 5.21 for QCIF and SIF respectively.

Sequence	**Processor**	**Chrominance Accelerator**	**Total Chrominance Prediction Time** (s)	**Avg. Time per CFrame** ($s/CFrame$)
Miss America	PowerPC	Disabled	5.314	0.018
		Enabled	5.621	0.019
	MicroBlaze	Disabled	31.716	0.106
		Enabled	6.983	0.023
Mobile	PowerPC	Disabled	26.261	0.060
		Enabled	21.882	0.050
	MicroBlaze	Disabled	140.998	0.320
		Enabled	27.058	0.061

Table 6.1: Chrominance Accelerator Execution Time Comparison

Communication costs associated with the chrominance prediction accelerator were evaluated using the test program written to verify functionality of the accelerator. The results of these tests are shown in Table 6.2. Normal operation indicates that the HW/SW system executes as it would when integrated into the JM Reference encoder copying the source frame to the accelerator, copying the resulting frames back from the accelerator, and polling for completion. The "No Results" column represents the execution time observed when polling was used to evaluate the status of the encoder and the source along with the resulting chrominance frames were not copied to or from the accelerator. The results reveal that over 80 percent of the total chrominance prediction execution time is devoted to copying the source frame and results to and from the BRAM of the accelerator and back to main memory. Note that the PowerPC clock rate is four times the MicroBlaze and PLB clocks. Thus the corresponding cycle counts for the PowerPC are substantially higher than the MicroBlaze cycle counts. As previously mentioned, the accelerator requires 281,148 cycles at 100 MHz to process one QCIF chrominance frame. Therefore, the additional polling overhead of the MicroBlaze and PowerPC is relatively minimal at less than 0.2 percent.

Processor	Normal Operation (CPU Cycles/CFrame)	No Results (CPU Cycles/CFrame)	Communication Percentage (%)
MicroBlaze	2,226,649	281,488	87.36
PowerPC	6,064,806	1,125,400	81.44

Table 6.2: Chrominance Communication Comparison Using QCIF Frame

6.2 Motion Estimation

Rate distortion curves comparing luminance PSNR and bit rate are shown in Figures 6.1, 6.2, 6.3, and 6.4. Note that the HA prefix in HA_HMDS and HA_FULL stands for Hardware Accelerator.

Each of the rate distortion curves demonstrates that HMDS yields performance which can compete with mainstream software motion estimation algorithms. The curves show that HMDS closely matches the PSNR of Fast Full search, but produces higher bit rates in line with the other algorithms. The Full search hardware accelerator performance closely matches the performance of Fast Full search, however produces slightly higher bit rates. The Full search accelerator does not match Fast Full search one to one due to the design of the added rate distortion unit. The rate distortion unit uses the same predicted motion vector for the current macroblock to predict the cost of each submacroblock rather than using updated motion vectors for prediction as the software full search does.

Figures 6.5 and 6.6 present the performance of HMDS in comparison with the other motion estimation algorithms on a frame by frame basis. Both figures show that HMDS achieves results which closely match existing algorithms. Specifically, HMDS consistently obtains PSNR values close to or exceeding Fast Full search PSNR values. Figure 6.6 reveals that the bit rates produced with HMDS are often on the high side.

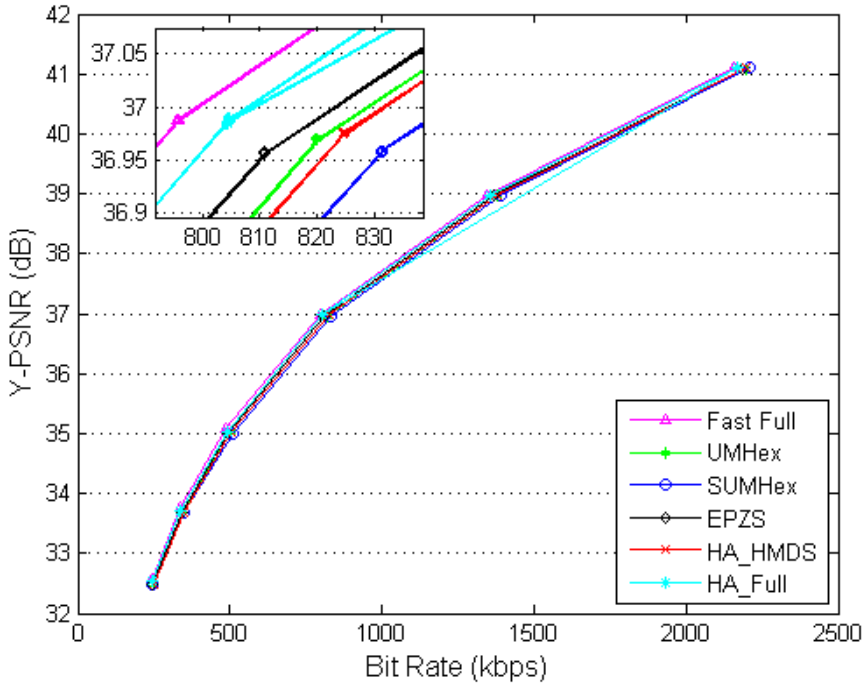


Figure 6.1: Rate Distortion Curve (Foreman, CIF, SR = 32, 1 ref frame, IPPP...)

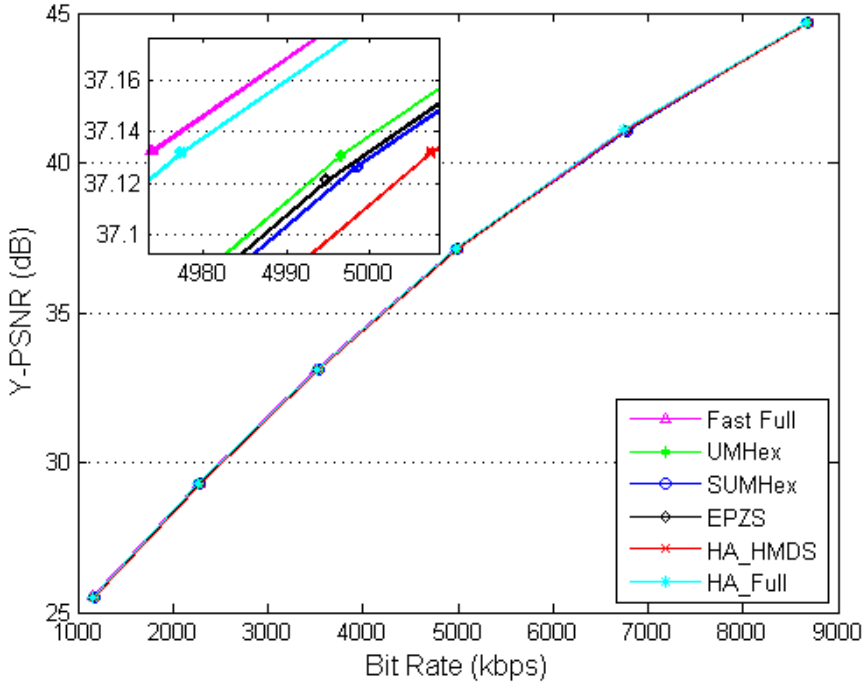


Figure 6.2: Rate Distortion Curve (Mobile, SIF, SR = 16, 1 ref frame, IPPP...)

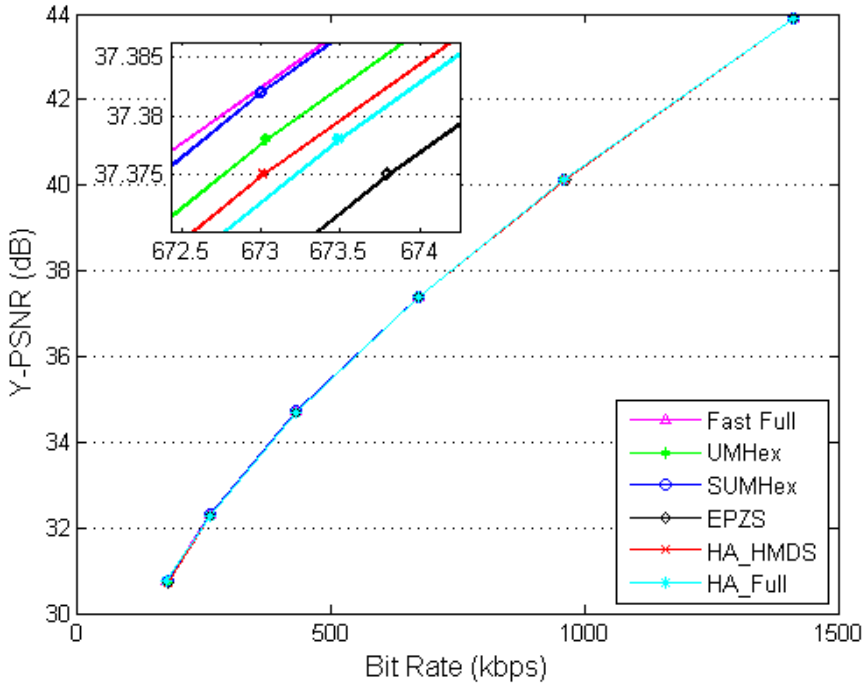


Figure 6.3: Rate Distortion Curve (Coastguard, QCIF, SR = 32, 1 ref frame, IPPP...)

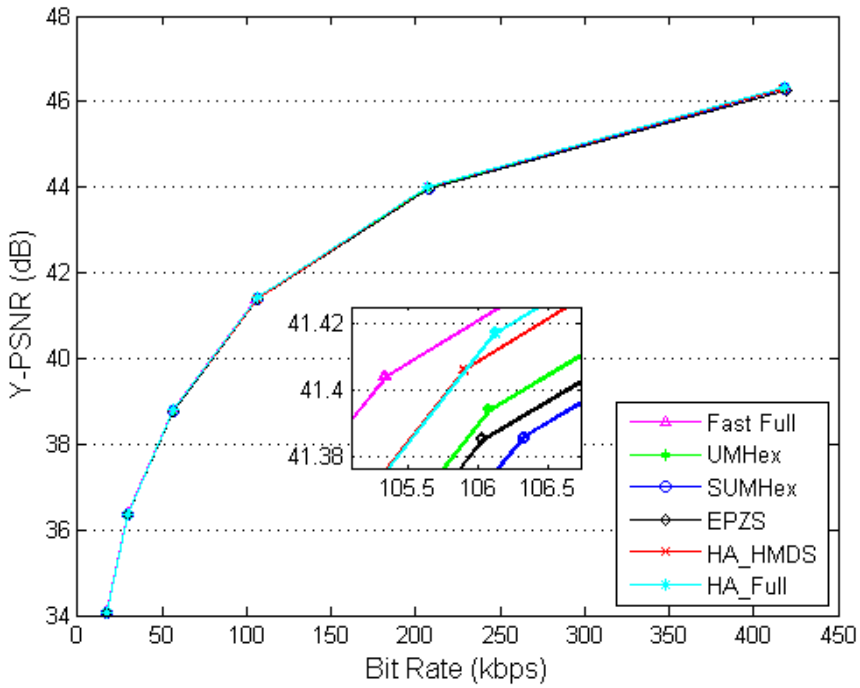


Figure 6.4: Rate Distortion Curve (Miss America, QCIF, SR = 16, 1 ref frame, IPPP...)

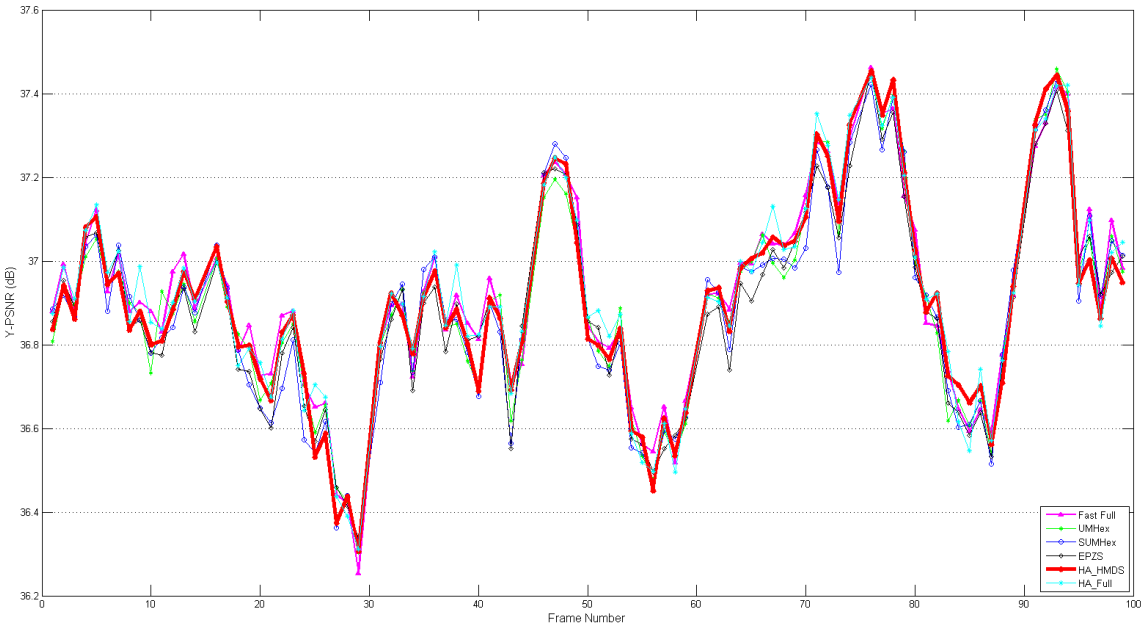


Figure 6.5: Frame By Frame Y-PSNR (Foreman, QCIF, QP=28, 1 ref frame, IPPP...)

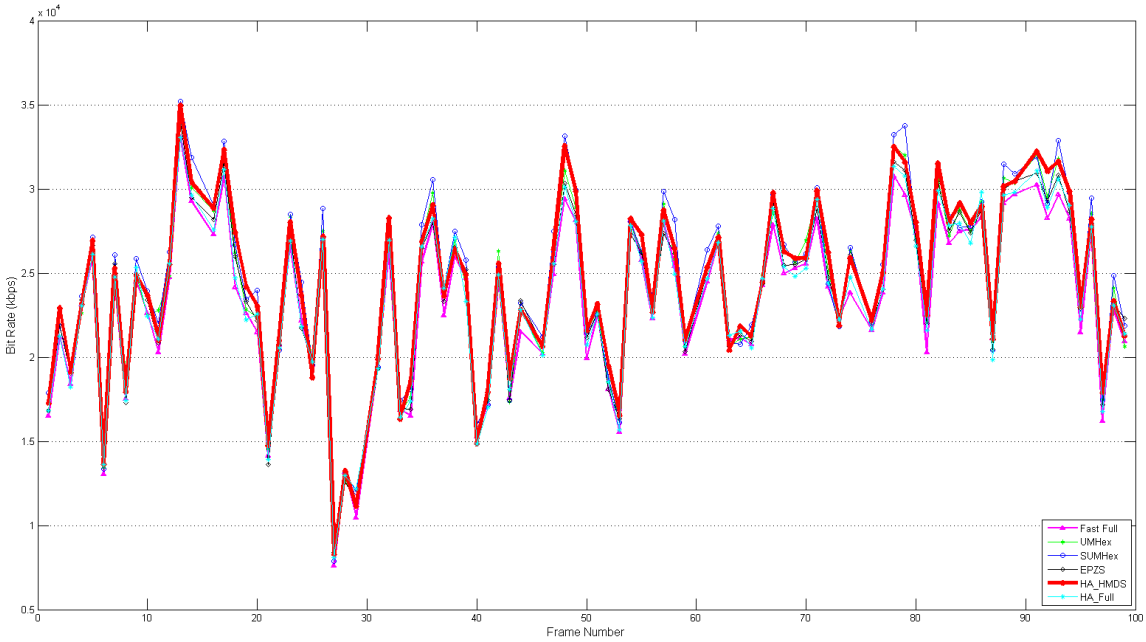


Figure 6.6: Frame By Frame Bit Rate (Foreman, QCIF, QP=28, 1 ref frame, IPPP...)

Table 6.3 shows the average luminance PSNR loss compared with Fast Full search for each of the motion estimation algorithms available in the modified hardware/software JM Reference baseline encoder. HMDS exhibits the smallest PSNR loss, excluding the full search hardware accelerator, for the majority of videos and produces higher PSNR values for three out of the sixteen sequences. As expected, the full search accelerator deviates the least overall from the PSNR of Fast Full search with less than .01 dB average PSNR loss across the majority of test sequences.

Sequence	**FF PSNR (dB)**	**UMHex**	**SUMHex**	**EPZS**	**HA_HMDS**	**HA_FULL**
foreman	36.4128	0.0373	0.0483	0.0478	0.0360	0.0185
mother	38.2707	0.0205	0.0158	0.0272	0.0102	0.0070
stefan	36.7942	-0.0015	0.0000	-0.0002	0.0398	-0.0007
flower	35.3207	0.0105	0.0078	0.0087	0.0025	0.0010
coast	36.5250	0.0008	-0.0002	0.0082	0.0055	0.0033
carphone	38.9673	0.0255	0.0240	0.0375	0.0095	0.0082
silent	37.8250	0.0040	0.0020	0.0070	0.0000	0.0017
suzie	38.9357	0.0123	0.0150	0.0197	0.0113	0.0105
mobile	35.1410	0.0088	0.0088	0.0138	0.0030	0.0025
tennis	35.5280	0.0127	0.0058	0.0132	0.0022	0.0050
water	35.3568	0.0088	0.0023	0.0117	0.0057	0.0045
contain	36.6967	0.0082	0.0080	0.0157	0.0047	0.0013
hall	37.9758	0.0093	0.0123	0.0165	0.0073	0.0008
salesman	36.2347	0.0048	0.0017	0.0030	0.0027	0.0003
news	37.3847	0.0083	0.0093	0.0093	0.0047	0.0020
miss	40.1680	0.0218	0.0223	0.0292	0.0122	0.0055

Table 6.3: Average Y-PSNR Loss Compared With Fast Full Search (dB)

Table 6.4 presents the average percentage bit rate increase for each of the motion estimation algorithms compared with Fast Full search. The increase in bit rate for HMDS most closely aligns with UMHex. Both UMHex and HMDS yield higher average bit rates than EPZS and lower bit rates than Simplified UMHex for the majority of sequences. This bit rate difference represents a significant performance increase considering that HMDS examines roughly half to a third of the locations of UMHex [11]. The full search hardware accelerator closely matches the bit rate of Fast Full search with all but one sequence producing less than one third of a percent increase in bit rate.

Sequence	**FF Bit Rate (kbit/s)**	**UMHex**	**SUMHex**	**EPZS**	**HA_HMDS**	**HA_FULL**
foreman	894.31	2.60	3.13	1.40	2.69	0.77
mother	329.06	1.09	1.01	0.78	1.13	0.40
stefan	5804.11	0.01	-0.00	-0.00	-0.76	0.00
flower	3101.72	0.39	0.24	0.15	0.29	0.07
coast	651.59	0.10	0.07	0.17	0.12	0.04
carphone	375.64	1.52	1.92	1.09	2.28	0.39
silent	241.91	0.59	0.43	0.43	1.02	0.31
suzie	292.91	0.80	0.95	0.56	0.49	0.20
mobile	4548.60	0.48	0.49	0.41	0.71	0.11
tennis	1823.78	1.70	1.26	0.82	1.93	0.24
water	1961.23	0.23	0.13	0.29	0.09	0.09
contain	858.55	0.02	0.04	0.16	0.04	0.04
hall	1139.85	0.21	0.22	0.13	0.24	0.04
salesman	171.90	0.14	0.20	0.07	0.16	0.11
news	188.17	0.25	0.16	0.09	0.26	0.07
miss	138.86	0.35	0.29	0.45	0.06	0.19

Table 6.4: Average Percentage Bit Rate Increase Compared With Fast Full Search (%)

Table 6.5 demonstrates the average speedup in comparison with UMHex for each motion estimation algorithm with the exception of Fast Full search. Note that Fast Full search results for the test suite were not collected due to the prohibitively long time required for completion on the available PowerPC and MicroBlaze processors. The HMDS accelerator exhibits speedups ranging from 3.448 to 17.024 over UMHex. On average, the HMDS hardware accelerator completes in roughly a third of the time required for the software EPZS algorithm. Despite examining significantly more locations than the other algorithms, the Full search accelerator manages to complete in a factor of ten times the duration of the UMHex algorithm.

Sequence	**UMHex Time (s)**	**SUMHex**	**EPZS**	**HA_HMDS**	**HA_FULL**
foreman	69.784	1.661	2.320	10.657	0.165
mother	61.362	1.476	1.755	6.348	0.097
stefan	115.220	1.188	1.702	17.024	1.311
flower	81.172	1.289	1.910	8.581	0.484
coast	52.426	1.774	3.863	15.170	0.226
carphone	35.023	1.552	2.640	9.947	0.152
silent	25.758	1.431	1.698	7.576	0.431
suzie	12.695	1.079	2.284	5.446	0.291
mobile	112.465	1.187	1.755	10.867	0.519
tennis	67.460	1.237	1.858	11.191	0.480
water	51.884	1.060	1.741	7.781	0.375
contain	81.854	1.400	1.873	3.448	0.319
hall	96.096	1.399	1.908	7.810	0.104
salesman	22.882	1.321	1.660	6.710	0.366
news	20.349	1.197	1.233	4.600	0.329
miss	15.250	1.451	1.999	6.286	0.095

Table 6.5: Average Speedup Compared With UMHex Search

Table 6.6 shows the communication costs associated with operating the motion estimation accelerator. These results were gathered using the motion estimation test program to isolate accelerator operation. The accelerator was given 44 candidate macroblocks to examine. In normal operation, macroblocks and motion vectors are sent over the PLB to populate the accelerator. When the accelerator completes, the optimal motion vectors and associated motion costs are sent back over the PLB to software. The stale data column omits sending and receiving data over the PLB. The only communication from software to the accelerator over the PLB in stale data mode is to modify the software accessible accelerator registers to reset, start, and poll for completion. Note that the number of cycles required for the accelerator to complete on the PowerPC is greater than for the MicroBlaze. This is because the PLB of the PowerPC operates at 100 MHz while the PowerPC clock is 400 MHz whereas both the CPU and PLB clock rate in the MicroBlaze configuration are 100 MHz. For both the PowerPC and the MicroBlaze configurations, PLB data communication requires approximately 95% of the total time for hardware motion acceleration. If the maximum motion estimation accelerator clock rate was used, this percentage would increase further. Note that according to eq. (4.7), the accelerator requires 341 cycles to process 44 candidate macroblocks. Thus, more than half the stale data cycle numbers are additional communication overhead due to polling and writing status data over the PLB.

In the worst case scenario, the results verify that 50,539 PowerPC cycles are required per macroblock. Thus, motion estimation using the designed HMDS accelerator should require 0.0125 seconds per QCIF frame or approximately 0.30 seconds per 24 frames for a total of 30% of the desired execution time.

Processor	**Normal Operation (CPU Cycles/Macroblock)**	**Stale Data (CPU Cycles/Macroblock)**	**Communication Percentage (%)**
MicroBlaze	33,223	781	97.65
PowerPC	50,539	2,726	94.61

Table 6.6: Motion Estimation Communication Comparison

Table 6.7 compares the designed motion estimation accelerator with the proposed architecture in [11] and previous accelerator designs. Given that one Virtex-5 CLB slice is roughly equivalent to two Virtex-2 Pro CLB slices, the designed accelerator closely matches the hardware cost and clock frequency of [11]. The removal of the Address Generation Unit and addition of the Rate Distortion Unit results in approximately the same hardware utilization. The slight decrease in clock frequency is likely due to the addition of the rate distortion unit and modification of the comparison units.

	Qiu et al	**Canals et al**	**Rahman et al**	**Roma et al**	**Modified SUMHex**	**HMDS [11]**	**This Work**
FPGA	Virtex-4	Virtex-4	Virtex-2	Virtex-E	Virtex-2-Pro	Virtex-2-Pro	Virtex-5
Algorithm	ACQPPS	Full Search	Full Search	Full Search	Modified SUMHex	HMDS	Modified HMDS
Search Range	± 16	± 48	± 4	± 16	± 16	± 16	N/A
Number of CLB slices	3.9K	12.5K	14.5K	14.7K	11.4K	7.8K	4.2K
Number of LUTs	3.2K	6.3K	28.5K	10.0K	18.7K	10.8K	10.3K
Clock Frequency (MHz)	60	100	149.2	76.1	145.2	246.5	227.8

Table 6.7: Comparison With Other Motion Estimation Architectures

6.3 Hardware/Software Encoder

The PowerPC configuration uses 7,799 slices, 16,505 slice registers, and 20,238 slice LUTs. Table 6.8 presents execution times using the PowerPC configuration for the Foreman (QP=28), Mobile (QP=30) and Miss America (QP=30) sequences with each of the non-full search algorithms.

Sequence	C Accel.	ME Alg.	Bit Rate (kbps)	Y-PSNR	Execution Time (s)	FPS
Foreman	Disabled	UMHex	820.03	36.969	282.617	0.354
		SUMHex	831.25	36.959	259.406	0.385
		EPZS	810.81	36.956	249.776	0.400
		HMDS	824.61	36.975	227.485	0.440
Mobile	Disabled	UMHex	3527.46	33.090	657.292	0.335
		SUMHex	3528.18	33.090	623.824	0.353
		EPZS	3521.27	33.080	595.825	0.369
		HMDS	3539.49	33.095	541.478	0.406
	Enabled	UMHex	3527.46	33.090	653.399	0.337
		SUMHex	3528.18	33.090	621.570	0.354
		EPZS	3521.27	33.080	589.728	0.373
		HMDS	3539.49	33.095	533.942	0.412
Miss America	Disabled	UMHex	57.02	38.783	94.900	1.581
		SUMHex	56.52	38.787	88.149	1.702
		EPZS	56.75	38.786	82.081	1.827
		HMDS	56.92	38.792	76.518	1.960
	Enabled	UMHex	57.02	38.783	95.127	1.577
		SUMHex	56.52	38.787	88.425	1.696
		EPZS	56.75	38.786	87.306	1.718
		HMDS	56.92	38.792	77.115	1.945

Table 6.8: HW/SW Encoder PowerPC Performance Comparison

The results demonstrate that HMDS alone yields a 8% average overall speedup over the fastest non-full motion estimation algorithm (EPZS) provided in the JM Reference encoder. Furthermore, HMDS obtains an average overall speedup of 24% over the slowest non-full motion estimation algorithm (UMHex). As previously mentioned, enabling the chrominance accelerator for QCIF sequences on the PowerPC hinders performance. However, enabling the chrominance accelerator with HMDS for the Mobile CIF

sequence yields speedups of 11.7% and 23.0% over the non-hardware accelerated EPZS and UMHex configurations respectively.

The MicroBlaze configuration is composed of 7,581 slices, 15,779 slice registers, and 19,431 slice LUTs. For further comparison, Table 6.9 presents execution times for the same encoding configurations as Table 6.8 running on the MicroBlaze rather than the PowerPC. Due to the PLB and MicroBlaze clocks running at the same rate, the accelerators have a greater impact on overall performance for the MicroBlaze than the PowerPC. HMDS without chrominance prediction acceleration yields a 11.8% average overall speedup over EPZS. Additionally, HMDS alone obtains an average overall speedup of 32.6% over UMHex. Enabling the chrominance accelerator alongside HMDS results in average overall speedups of 17.7% and 38.4% for EPZS and UMHex respectively.

Sequence	**C Accel.**	**ME Alg.**	**Bit Rate (kbps)**	**Y-PSNR**	**Execution Time (s)**	**FPS**
Foreman	Disabled	UMHex	820.02	36.970	1850.690	0.054
		SUMHex	831.25	36.959	1589.496	0.063
		EPZS	809.21	36.954	1503.616	0.067
		HMDS	824.61	36.975	1327.377	0.075
Mobile	Disabled	UMHex	3526.45	33.091	4004.713	0.055
		SUMHex	3528.18	33.090	3704.801	0.059
		EPZS	3522.09	33.080	3478.410	0.063
		HMDS	3539.49	33.095	3059.425	0.072
	Enabled	UMHex	3526.45	33.091	3880.190	0.057
		SUMHex	3528.18	33.090	3590.892	0.061
		EPZS	3522.09	33.080	3364.505	0.065
		HMDS	3539.49	33.095	2938.457	0.075
Miss America	Disabled	UMHex	57.02	38.783	606.498	0.247
		SUMHex	56.52	38.787	527.262	0.284
		EPZS	56.77	38.784	513.685	0.292
		HMDS	56.92	38.792	461.763	0.325
	Enabled	UMHex	57.02	38.783	581.763	0.258
		SUMHex	56.52	38.787	502.526	0.298
		EPZS	56.77	38.784	489.120	0.307
		HMDS	56.92	38.792	437.059	0.343

Table 6.9: HW/SW Encoder MicroBlaze Performance Comparison

Chapter 7

Conclusions

The chrominance prediction accelerator exhibited performance slower than software prediction when integrated into the PowerPC encoder for QCIF frames. Results indicated that 80% of the time required for the accelerator to process a frame was spent sending/receiving data to and from software. To truly reap the benefits of a chrominance accelerator, hardware needs to be able to write resulting data directly to a permanent memory location rather than to BRAM for temporary storage.

The architecture proposed in [11] was validated for performance amongst competing software motion estimation algorithms. In addition, HMDS was confirmed to yield high quality encoded video at the expense of higher bit rates. Furthermore, the algorithm was modified to yield higher quality results through the addition of two motion vector predictors and in-accelerator motion vector encoding cost estimates. The proposed architecture was also modified to increase motion estimation flexibility in a hardware/software system. The resulting full search hardware accelerator was found to yield results closely matching full search at a considerable reduction in processing time. Like the chrominance prediction accelerator, the motion estimation accelerator would benefit from writing directly to main memory rather than transferring data over the PLB to main memory. About 94% of the time required to process one macroblock using HMDS is spent transferring data between the accelerator and main memory, significantly hindering the optimal speedup of the accelerator.

The resulting hardware/software encoder was found to add flexibility, decrease overall execution time, and retain quality in comparison the original JM Reference encoder. All configurable parameters in the modified

JM Reference Baseline encoder were preserved and new parameters were added to enable hardware accelerators. By enabling the HMDS motion accelerator, the PowerPC configuration was found to encode 150 frames of the Miss America QCIF test sequence at 1.960 FPS with a luminance PSNR of 38.792 dB and a bit rate of 56.92 kbps. This performance is an improvement over the software only encoder, but is still nowhere near the goal of real-time encoding at 24 FPS. The results found that the designed HMDS accelerator is capable of producing 24 QCIF in 0.30 seconds. Therefore, real-time encoding could be achieved if the software encoder could encode 24 QCIF frames in 0.70 seconds excluding motion estimation. As the results demonstrate, the JM Reference encoder is not capable of meeting this constraint. However, the findings in [8] suggest that the open source x264 software encoder performs on average 50x faster than the JM Reference encoder. If the HMDS accelerator was ported to the x264 encoder, real-time encoding at 24 FPS should be possible.

7.1 Future Work

This thesis confirmed that hardware accelerators can decrease encoding time while maintaining the flexibility of software. However, the addition of a chrominance prediction accelerator and motion estimation accelerator resulted in substantial communication overhead between the accelerators and software. Thus, future work could be devoted to making several improvements to reduce communication costs.

The design of the chrominance prediction accelerator was adjusted to fit the hardware constraints of the XC5VFX70T. If more dual-port BRAM was available, the accelerator could be designed to compute predicted frames in parallel. Computing seven diagonal, one horizontal, and one vertical frame at a time would likely provide a proper balance between hardware costs and execution time. In addition, having BRAM large enough to hold all 128 predicted frames would remove the need to transfer resulting data over the PLB to DDR2, thus further decreasing the time required for completion.

Overall, creating hardware accelerators for the JM Reference encoder proved successful in decreasing encoding time while maintaining the flexibility of software. However, the question of whether the additional hardware costs merit the decrease encoding time arises. As previously mentioned, the bulk of time expended on hardware acceleration is spent on transferring data between software and hardware which verifies the observations in [12]. To obtain optimal results, main memory must be directly accessible to both hardware and software. However, even if all of main memory could be shared between the CPU and the associated hardware accelerators, significant changes to the architecture of the software encoder would have to be made. The JM Reference encoder is structured with numerous pointer levels ranging up to seven levels deep and contains an excessive number of data structures for storing statistics and providing interoperability with numerous profiles and component algorithms. To create a flexible hardware/software H.264 encoder capable of encoding low bit rate video sequences in real time at frame rates in excess of 24 FPS, the architecture would require quick access to main memory from both hardware and software and a simplistic memory map such that hardware can access macroblocks without the overhead of dereferencing multiple pointers.

In addition to tweaking the accelerators designed in this thesis, additional accelerators could be designed to target computationally intensive software bottlenecks. This text noted that luminance prediction, like chrominance prediction, represents a critical software bottleneck. Portions of the chrominance prediction accelerator could be directly ported to a luminance prediction accelerator given that four of the thirteen luminance prediction methods are the same as those for chrominance. Furthermore, although the DCT and Hadamard transforms did not top the list of methods for software bottlenecks, their simplistic, fixed size computation make them ideal for hardware acceleration.

To increase flexibility in low bit rate environments, RoI encoding could be added either in software or hardware. The addition of RoI encoding would allow for frame redundancy to reduce the likelihood of decoding errors in channel constrained networks. Moreover, RoI encoding would maintain high quality foreground regions of interest while allowing the background quality to dwindle to yield a perceived high quality video in a low bit rate environment.

Bibliography

[1] Zhibo Chen, Peng Zhou, and Yun He. Fast integer pel and fractional pel motion estimation for jvt. 2002.

[2] Zhibo Chen, Peng Zhou, and Yun He. Fast motion estimation for jvt. 2003.

[3] Cludio Machado Diniz, Bruno Zatt, Luciano Agostin, Altamiro Susin, and Sergio Bampi. A real time H.264/AVC intra frame prediction hardware architecture for HDTV 1080P video. *ICME 2009*, pages 1138 – 1141, 2009.

[4] Tony Gladvin George. Video coding basics, 2007.

[5] Wenwei He and Yuling Zhang. Improved hexagon-based searching algorithm for fast motion estimation. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*, pages 1 –3, sept. 2010.

[6] Chao-Yang Kao, Cheng-Long Wu, and Youn-Long Lin. A high-performance three-engine architecture for H.264/AVC fractional motion estimation. *IEEE J. VLSI*, 18(4):662–666, 2010.

[7] Vincenzo Liguori. The power and bandwidth advantage of an H.264 IP core with 8-16:1 compressed reference frame store, March 2011.

[8] Loren Merritt and Rahul Vanam. X264: A high performance h.264/avc

encoder. *online httpneuron2 netlibraryavcoverview pdf*, pages 1–15, 2004.

[9] Theepan Moorthy and Andy Ye. A scalable computing and memory architecture for variable block size motion estimation on field-programmable gate arrays. In *Proc. Int. Conf. Field Programmable Logic and Applications FPL 2008*, pages 83–88, 2008.

[10] Obianuju Ndili and Tokunbo Ogunfunmi. Hardware-oriented modified diamond search for motion estimation in h.246/avc. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 749 –752, sept. 2010.

[11] Obianuju Ndili and Tokunbo Ogunfunmi. Algorithm and architecture co-design of hardware-oriented, modified diamond search for fast motion estimation in h.264/avc. *Circuits and Systems for Video Technology, IEEE Transactions on*, PP(99):1, 2011.

[12] Stijn Notebaert and Jan De Cock. Hardware/software co-design of the H.264/AVC standard, December 2004.

[13] Antonio Ortega and Kannan Ramchandran. Rate-distortion methods for image and video compression. *Signal Processing Magazine, IEEE*, 15(6):23 –50, nov 1998.

[14] Muhsen Owaida, Maria Koziri, Ioannis Katsavounidis, and George Stamoulis. A high performance and low power hardware architecture for the transform & quantization stages in H.264. In *Proc. IEEE Int. Conf. Multimedia and Expo ICME 2009*, pages 1102–1105, 2009.

[15] Iain Richardson. *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*. Wiley, 2003.

[16] Ralf Schafer, Thomas Wiegand, and Heiko Schwarz. The emerging h.264/avc standard. In *EBU Technical Review*, January 2003.

[17] Timothy Sperr. Investigating low-bitrate, low-complexity H.264 region of interest techniques in error-prone environments. 2011.

[18] Alexis Tourapis. Fast me in the jm reference software, July 2005.

[19] Alexis Tourapis, Oscar Au, and Ming Liou. Highly efficient predictive zonal algorithms for fast block-matching motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 12(10):934 – 947, oct 2002.

[20] Stefan Winkler and Praveen Mohandas. The evolution of video quality measurement: From psnr to hybrid metrics. *Broadcasting, IEEE Transactions on*, 54(3):660 –668, September 2008.

[21] Xilinx. *Virtex-5 Family Overview*, v5.0 edition, February 2009.

[22] Xilinx. *Embedded Processor Block in Virtex-5 FPGAs Reference Guide*, v1.8 edition, February 2010.

[23] Xilinx. *LogiCORE IP Processor Local Bus (PLB) v4.6*, v1.05a edition, September 2010.

[24] Xilinx. *MicroBlaze Processor Reference Guide*, v12.0 edition, March 2011.

[25] Jianfeng Xu, Zhibo Chen, and Yun He. Efficient fast me predictions and early-termination strategy based on h.264 statistical characters. In *Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, volume 1, pages 218 – 222 Vol.1, dec. 2003.

[26] Jianfeng Xu, Ping Yang, and Yun He. Modification of fast motion estimation, December 2003.